

INTERNATIONAL  
STANDARD

**ISO/IEC/  
IEEE  
9945**

First edition  
2009-09-15

---

---

**Information technology — Portable  
Operating System Interface (POSIX®)  
Base Specifications, Issue 7**

*Technologies de l'information — Spécifications de base de l'interface  
pour la portabilité des systèmes (POSIX®), Issue 7*

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009



Reference number  
ISO/IEC/IEEE 9945:2009(E)



Copyright © 2001-2008, IEEE and The Open Group.  
All rights reserved

**PDF disclaimer**

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. Neither the ISO Central Secretariat nor IEEE accepts any liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies and IEEE members. In the unlikely event that a problem relating to it is found, please inform the ISO Central Secretariat or IEEE at the address given below.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009



**COPYRIGHT PROTECTED DOCUMENT**

© IEEE 2001-2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO or IEEE at the respective address below.

ISO copyright office  
Case postale 56 • CH-1211 Geneva 20  
Tel. + 41 22 749 01 11  
Fax + 41 22 749 09 47  
E-mail [copyright@iso.org](mailto:copyright@iso.org)  
Web [www.iso.org](http://www.iso.org)

Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York • NY 10016-5997, USA  
E-mail [stds.ipr@ieee.org](mailto:stds.ipr@ieee.org)  
Web [www.ieee.org](http://www.ieee.org)

Published in Switzerland

## Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

The main task of ISO/IEC JTC 1 is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from ISO or the IEEE Standards Association.

ISO/IEC/IEEE 9945 was prepared by The Open Group (as The Open Group Technical Standard Base Specifications, Issue 7) and the Portable Applications Standards Committee of the Computer Society of the IEEE (as IEEE Std 1003.1™-2008). It was adopted by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*, in parallel with its approval by the ISO/IEC national bodies, under the “fast-track procedure” defined in the Partner Standards Development Organization cooperation agreement between ISO and IEEE. IEEE is responsible for the maintenance of this document with participation and input from ISO/IEC national bodies.

This first edition of ISO/IEC/IEEE 9945 cancels and replaces ISO/IEC 9945-1:2003, ISO/IEC 9945-2:2003, ISO/IEC 9945-3:2003 and ISO/IEC 9945-4:2003, which have been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 9945-1:2003/Cor.1:2004, ISO/IEC 9945-2:2003/Cor.1:2004, ISO/IEC 9945-3:2003/Cor.1:2004 and ISO/IEC 9945-4:2003/Cor.1:2004.

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009



# Standard for Information Technology— Portable Operating System Interface (POSIX®)

## Base Specifications, Issue 7

---

### IEEE Computer Society

Sponsored by the  
Portable Applications Standards Committee  
and  
The Open Group

---

IEEE  
3 Park Avenue  
New York, NY 10016-5997, USA  
1 December 2008

**IEEE Std 1003.1™-2008**  
(Revision of  
IEEE Std 1003.1-2004)

TM  
1003.1

STANDARDSISO.COM :: Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

IEEE Std 1003.1™ -2008  
(Revision of  
IEEE Std 1003.1-2004)

The Open Group Technical Standard  
Base Specifications, Issue 7

# Standard for Information Technology— Portable Operating System Interface (POSIX®) Base Specifications, Issue 7

Sponsor

Portable Applications Standards Committee

of the

IEEE Computer Society

and

The Open Group

Approved 26 September 2008

IEEE-SA Standards Board

Approved 24 July 2008

The Open Group



## Abstract

POSIX.1-2008 is simultaneously IEEE Std 1003.1™-2008 and The Open Group Technical Standard Base Specifications, Issue 7.

POSIX.1-2008 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. POSIX.1-2008 is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-2008 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of POSIX.1-2008:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2008 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

## Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX®), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

---

The Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, USA

The Open Group  
Thames Tower, Station Road, Reading, Berkshire, RG1 1LX, U.K.

Copyright © 2008 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group  
All rights reserved.

Published 1 December 2008 by the IEEE. Printed in the United States of America by the IEEE.

PDF: ISBN 978-0-7381-5798-6 STD95820  
CDROM: ISBN 978-0-7381-5799-3 STDCD95820

Published 1 December 2008 by The Open Group. Printed in the United Kingdom by The Open Group.

Doc. Number: C082  
ISBN: 1-931624-79-8

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher. Permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests should be sent by email to [austin-group-permissions@opengroup.org](mailto:austin-group-permissions@opengroup.org).*

*This standard has been prepared by the Austin Group. Feedback relating to the material contained within this standard may be submitted by using the Austin Group web site at [www.opengroup.org/austin/defectform.html](http://www.opengroup.org/austin/defectform.html).*

## IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "AS IS".

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE Standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE.<sup>A</sup> Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board  
445 Hoes Lane  
Piscataway, NJ 08854  
USA

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

<sup>A</sup> For this standard please send comments via the Austin Group, as indicated on page ii.

## The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX® certification.

Further information on The Open Group is available at [www.opengroup.org](http://www.opengroup.org).

The Open Group has over 20 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at [www.opengroup.org/bookstore](http://www.opengroup.org/bookstore).

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at [www.opengroup.org/corrigenda](http://www.opengroup.org/corrigenda).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Introduction

This introduction is not part of POSIX.1-2008, Standard for Information Technology – Portable Operating System Interface (POSIX).

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.<sup>B</sup>

The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group is to develop and maintain the core open systems interfaces that are the POSIX<sup>®</sup> 1003.1 (and former 1003.2) standards, ISO/IEC 9945, and the core of the Single UNIX Specification.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see [www.opengroup.org/austin](http://www.opengroup.org/austin).

## Background

The developers of POSIX.1-2008 represent a cross-section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, POSIX.1-2008 describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application developers to write portable applications – it was developed with that goal in mind – it has been designated POSIX<sup>C</sup>, an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

## Audience

The intended audience for POSIX.1-2008 is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

- Persons buying hardware and software systems
- Persons managing companies that are deciding on future corporate computing directions
- Persons implementing operating systems, and especially
- Persons developing applications where portability is an objective

<sup>B</sup> The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

<sup>C</sup> The Name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

**Purpose**

Several principles guided the development of POSIX.1-2008:

- Application-Oriented – The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. POSIX.1-2008 codifies the common, existing definition of the UNIX system.
- Interface, Not Implementation – POSIX.1-2008 defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.
- Source, Not Object, Portability – POSIX.1-2008 has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. POSIX.1-2008 does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.
- The C Language – The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.
- No Superuser, No System Administration – There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in POSIX.1-2008. POSIX.1-2008 is also not concerned with hardware constraints or system maintenance.
- Minimal Interface, Minimally Defined – In keeping with the historical design principles of the UNIX system, the mandatory core facilities of POSIX.1-2008 have been kept as minimal as possible. Additional capabilities have been added as optional extensions.
- Broadly Implementable – The developers of POSIX.1-2008 endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:
  - All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
  - Compatible systems that are not derived from the original UNIX system code
  - Emulations hosted on entirely different operating systems
  - Networked systems
  - Distributed systems
  - Systems running on a broad range of hardware

No direct references to this goal appear in POSIX.1-2008, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations – When the original version – IEEE Std 1003.1-1988 – was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and IEEE Std 1003.1-2001 and its technical corrigenda have consolidated this consensus, and this version reflects the significantly increased level of consensus arrived at since the original versions. The authors of the original versions tried, as much as possible, to follow the principles below when creating new specifications:

- By standardizing an interface like one in an historical implementation; for example, directories
- By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
- By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

POSIX.1-2008 is specifically not a codification of a particular vendor's product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of POSIX.1-2008. These need to be used with careful management to be able to adapt to future extensions of POSIX.1-2008 and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code – A goal of POSIX.1-2008 was to minimize additional work for application developers. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

### POSIX.1-2008

POSIX.1-2008 defines the Portable Operating System Interface (POSIX) requirements and consists of the following topics arranged as a series of volumes within the standard:

- Base Definitions
- System Interfaces
- Shell and Utilities
- Rationale (Informative)

### Base Definitions

The Base Definitions volume provides common definitions for this standard, therefore readers should be familiar with it before using the other volumes.

This volume is structured as follows:

- Chapter 1 is an introduction.
- Chapter 2 defines the conformance requirements.
- Chapter 3 defines general terms used.
- Chapter 4 describes general concepts used.
- Chapter 5 describes the notation used to specify file input and output formats in this volume and the Shell and Utilities volume.
- Chapter 6 describes the portable character set and the process of character set definition.
- Chapter 7 describes the syntax for defining internationalization locales as well as the POSIX locale provided on all systems.
- Chapter 8 describes the use of environment variables for internationalization and other purposes.
- Chapter 9 describes the syntax of pattern matching using regular expressions employed by many utilities and matched by the *regcomp()* and *regexexec()* functions.
- Chapter 10 describes files and devices found on all systems.
- Chapter 11 describes the asynchronous terminal interface for many of the functions in the System Interfaces volume and the *stty* utility in the Shell and Utilities volume.
- Chapter 12 describes the policies for command line argument construction and parsing.
- Chapter 13 defines the contents of headers which declare the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by the System Interfaces volume.

Comprehensive references are available in the index.

### System Interfaces

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of this volume.
- Chapter 3 defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the index.

### Shell and Utilities

The Shell and Utilities volume describes the commands and utilities offered to application programs on POSIX-conformant systems. Readers are expected to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards. It also describes the defaults used by the utility descriptions.
- Chapter 2 describes the command language used in POSIX-conformant systems, and special built-in utilities.
- Chapter 3 describes a set of services and utilities that are implemented on systems supporting the Batch Environment Services and Utilities option.
- Chapter 4 consists of reference pages for all utilities, other than the special built-in utilities described in Chapter 2, available on POSIX-conformant systems.

Comprehensive references are available in the index.

### Rationale (Informative)

The Rationale volume is published to assist in the process of review. It contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of POSIX.1-2008 that may not be immediately apparent.

This volume is organized in parallel to the normative volumes of this standard, with a separate part for each of the three normative volumes.

Within this volume, the following terms are used:

- Base standard – The portions of POSIX.1-2008 that are not optional, equivalent to the definitions of *classic* POSIX.1 and POSIX.2.
- POSIX.0 – Although this term is not used in the normative text of POSIX.1-2008, it is used in this volume to refer to IEEE Std 1003.0™-1995.
- POSIX.1b – Although this term is not used in the normative text of POSIX.1-2008, it is used in this volume to refer to the elements of the POSIX Realtime Extension amendment. (This was earlier referred to as POSIX.4 during the standard development process.)
- POSIX.1c – Although this term is not used in the normative text of POSIX.1-2008, it is used in this volume to refer to the POSIX Threads Extension amendment. (This was earlier referred to as POSIX.4a during the standard development process.)
- Standard developers – The individuals and companies in the development organizations responsible for POSIX.1-2008: the IEEE P1003.1 working groups, The Open Group Base working group, advised by the hundreds of individual technical experts who balloted the draft standards within the Austin Group, and the member bodies and technical experts of ISO/IEC JTC 1/SC 22.
- XSI option – The portions of POSIX.1-2008 addressing the extension added for support of the Single UNIX Specification.

### Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as POSIX.1-2008, which is technically identical to The Open Group Base Specifications, Issue 7.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in POSIX.1-2008.

Reference	Example	Notes
C-Language Data Structure	<b>aiocb</b>	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	<b>long</b>	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	<b>return</b>	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	NET_ADDRSTRLEN	
C-Language Preprocessing Directive	<b>#define</b>	
Commands within a Utility	<b>a, c</b>	
Conversion Specifier, Specifier/Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	<b>Hello, World</b>	
Filename	<i>/tmp</i>	
Literal Character	'c', '\r'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[ ]	
Parameter	<directory pathname>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	
User Input and Example Code	echo Hello, World	5
Utility Name	<i>awk</i>	
Utility Operand	<i>file_name</i>	
Utility Option	<b>-c</b>	
Utility Option with Option-Argument	<b>-w width</b>	

Note that:

- Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf()* and *fscanf()* formatting functions.
- Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. The literal characters <apostrophe> (also known as single-quote) and <backslash> are either shown as the C constants '\ ' and '\\ ', respectively, or as the special characters <apostrophe>, single-quote, and <backslash> depending on context.
- The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.
- Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.

5. Brackets shown in this font, " [ ] ", are part of the syntax and do not indicate optional items. In syntax the ' | ' symbol is used to separate alternatives, and ellipses ( " . . . " ) are used to show that additional arguments are optional.

Shading is used to identify extensions and options.

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

1.  $(a,b)$  means the range of all values from  $a$  to  $b$ , including neither  $a$  nor  $b$ .
2.  $[a,b]$  means the range of all values from  $a$  to  $b$ , including  $a$  and  $b$ .
3.  $[a,b)$  means the range of all values from  $a$  to  $b$ , including  $a$ , but not  $b$ .
4.  $(a,b]$  means the range of all values from  $a$  to  $b$ , including  $b$ , but not  $a$ .

**Note:** A symbolic limit beginning with POSIX is treated differently, depending on context. In a C-language header, the symbol `POSIXstring` (where *string* may contain underscores) is represented by the C identifier `_POSIXstring`, with a leading underscore required to prevent ISO C standard name space pollution. However, in other contexts, such as languages other than C, the leading underscore is not used because this requirement does not exist.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Notice to Users

### Laws and Regulations

Users of this document should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE and The Open Group do not, by the publication of standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

### Copyrights

This document is copyrighted by the IEEE and The Open Group. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE and The Open Group do not waive any rights in copyright to this document.

### Updating of IEEE Documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association web site at [ieeexplore.ieee.org/xpl/standards.jsp](http://ieeexplore.ieee.org/xpl/standards.jsp), or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at [standards.ieee.org](http://standards.ieee.org).

### Errata

Errata, if any, for this and all other standards can be accessed at the following web site: [standards.ieee.org/reading/ieee/updates/errata](http://standards.ieee.org/reading/ieee/updates/errata). Users are encouraged to check this URL for errata periodically.

### Feedback

POSIX.1-2008 has been prepared by the Austin Group. Feedback relating to the material contained in POSIX.1-2008 may be submitted using the Austin Group web site at [www.opengroup.org/austin/defectform.html](http://www.opengroup.org/austin/defectform.html).

### Interpretations

Current interpretations can be accessed at the following web site: [standards.ieee.org/reading/ieee/interp](http://standards.ieee.org/reading/ieee/interp).

### Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses. Other Essential Patent Claims may exist for which a statement of assurance has not been received. The IEEE and The Open Group are not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions are reasonable or non-discriminatory. Further information may be obtained from the IEEE Standards Association and The Open Group.

# Contents

<b>Volume</b>	<b>1</b>	<b>Base Definitions, Issue 7</b> .....	<b>1</b>
<b>Chapter</b>	<b>1</b>	<b>Introduction</b> .....	<b>3</b>
	1.1	Scope .....	3
	1.2	Conformance.....	4
	1.3	Normative References .....	4
	1.4	Change History .....	5
	1.5	Terminology .....	5
	1.6	Definitions and Concepts.....	6
	1.7	Portability .....	6
	1.7.1	Codes .....	7
	1.7.2	Margin Code Notation .....	13
<b>Chapter</b>	<b>2</b>	<b>Conformance</b> .....	<b>15</b>
	2.1	Implementation Conformance .....	15
	2.1.1	Requirements .....	15
	2.1.2	Documentation .....	16
	2.1.3	POSIX Conformance.....	16
	2.1.4	XSI Conformance.....	19
	2.1.5	Option Groups.....	20
	2.1.6	Options .....	26
	2.2	Application Conformance.....	29
	2.2.1	Strictly Conforming POSIX Application.....	29
	2.2.2	Conforming POSIX Application .....	30
	2.2.3	Conforming POSIX Application Using Extensions.....	30
	2.2.4	Strictly Conforming XSI Application .....	30
	2.2.5	Conforming XSI Application Using Extensions .....	31
	2.3	Language-Dependent Services for the C Programming Language .....	31
	2.4	Other Language-Related Specifications.....	31
<b>Chapter</b>	<b>3</b>	<b>Definitions</b> .....	<b>33</b>
	3.1	Abortive Release.....	33
	3.2	Absolute Pathname.....	33
	3.3	Access Mode .....	33
	3.4	Additional File Access Control Mechanism.....	33
	3.5	Address Space.....	33
	3.6	Advisory Information.....	34
	3.7	Affirmative Response .....	34
	3.8	Alert .....	34
	3.9	Alert Character (<alert>).....	34
	3.10	Alias Name.....	34
	3.11	Alignment .....	35
	3.12	Alternate File Access Control Mechanism .....	35

## Contents

3.13	Alternate Signal Stack.....	35
3.14	Ancillary Data.....	35
3.15	Angle Brackets.....	35
3.16	Apostrophe Character (<apostrophe>).....	35
3.17	Application.....	35
3.18	Application Address.....	36
3.19	Application Program Interface (API).....	36
3.20	Appropriate Privileges.....	36
3.21	Argument.....	36
3.22	Arm (a Timer).....	36
3.23	Asterisk Character (<asterisk>).....	36
3.24	Async-Cancel-Safe Function.....	36
3.25	Asynchronous Events.....	37
3.26	Asynchronous Input and Output.....	37
3.27	Async-Signal-Safe Function.....	37
3.28	Asynchronously-Generated Signal.....	37
3.29	Asynchronous I/O Completion.....	37
3.30	Asynchronous I/O Operation.....	37
3.31	Authentication.....	37
3.32	Authorization.....	38
3.33	Background Job.....	38
3.34	Background Process.....	38
3.35	Background Process Group (or Background Job).....	38
3.36	Backquote Character.....	38
3.37	Backslash Character (<backslash>).....	38
3.38	Backspace Character (<backspace>).....	38
3.39	Barrier.....	39
3.40	Basename.....	39
3.41	Basic Regular Expression (BRE).....	39
3.42	Batch Access List.....	39
3.43	Batch Administrator.....	39
3.44	Batch Client.....	39
3.45	Batch Destination.....	40
3.46	Batch Destination Identifier.....	40
3.47	Batch Directive.....	40
3.48	Batch Job.....	40
3.49	Batch Job Attribute.....	40
3.50	Batch Job Identifier.....	40
3.51	Batch Job Name.....	41
3.52	Batch Job Owner.....	41
3.53	Batch Job Priority.....	41
3.54	Batch Job State.....	41
3.55	Batch Name Service.....	41
3.56	Batch Name Space.....	41
3.57	Batch Node.....	42
3.58	Batch Operator.....	42
3.59	Batch Queue.....	42
3.60	Batch Queue Attribute.....	42
3.61	Batch Queue Position.....	42
3.62	Batch Queue Priority.....	42
3.63	Batch Rerunability.....	43
3.64	Batch Restart.....	43

Contents

3.65	Batch Server .....	43
3.66	Batch Server Name.....	43
3.67	Batch Service .....	43
3.68	Batch Service Request.....	43
3.69	Batch Submission .....	43
3.70	Batch System .....	44
3.71	Batch Target User .....	44
3.72	Batch User .....	44
3.73	Bind .....	44
3.74	Blank Character (<blank>).....	44
3.75	Blank Line.....	44
3.76	Blocked Process (or Thread) .....	44
3.77	Blocking .....	44
3.78	Block-Mode Terminal .....	45
3.79	Block Special File.....	45
3.80	Braces .....	45
3.81	Brackets.....	45
3.82	Broadcast .....	45
3.83	Built-In Utility (or Built-In).....	46
3.84	Byte.....	46
3.85	Byte Input/Output Functions .....	46
3.86	Carriage-Return Character (<carriage-return>) .....	46
3.87	Character .....	47
3.88	Character Array .....	47
3.89	Character Class.....	47
3.90	Character Set.....	47
3.91	Character Special File .....	47
3.92	Character String.....	47
3.93	Child Process .....	48
3.94	Circumflex Character (<circumflex>).....	48
3.95	Clock .....	48
3.96	Clock Jump.....	48
3.97	Clock Tick.....	48
3.98	Coded Character Set .....	48
3.99	Codeset .....	49
3.100	Collating Element.....	49
3.101	Collation .....	49
3.102	Collation Sequence.....	49
3.103	Column Position.....	50
3.104	Command.....	50
3.105	Command Language Interpreter .....	50
3.106	Composite Graphic Symbol.....	50
3.107	Condition Variable .....	50
3.108	Connected Socket.....	51
3.109	Connection .....	51
3.110	Connection Mode.....	51
3.111	Connectionless Mode .....	51
3.112	Control Character.....	51
3.113	Control Operator.....	51
3.114	Controlling Process .....	51
3.115	Controlling Terminal .....	52
3.116	Conversion Descriptor .....	52

## Contents

3.117	Core File.....	52
3.118	CPU Time (Execution Time) .....	52
3.119	CPU-Time Clock.....	52
3.120	CPU-Time Timer.....	52
3.121	Current Job.....	52
3.122	Current Working Directory.....	53
3.123	Cursor Position.....	53
3.124	Datagram.....	53
3.125	Data Segment.....	53
3.126	Deferred Batch Service .....	53
3.127	Device .....	53
3.128	Device ID.....	53
3.129	Directory.....	53
3.130	Directory Entry (or Link) .....	53
3.131	Directory Stream .....	54
3.132	Disarm (a Timer) .....	54
3.133	Display.....	54
3.134	Display Line.....	54
3.135	Dollar-Sign Character (<dollar-sign>).....	54
3.136	Dot.....	54
3.137	Dot-Dot.....	55
3.138	Double-Quote Character.....	55
3.139	Downshifting.....	55
3.140	Driver.....	55
3.141	Effective Group ID .....	55
3.142	Effective User ID.....	55
3.143	Eight-Bit Transparency .....	55
3.144	Empty Directory.....	56
3.145	Empty Line.....	56
3.146	Empty String (or Null String).....	56
3.147	Empty Wide-Character String.....	56
3.148	Encoding Rule .....	56
3.149	Entire Regular Expression.....	56
3.150	Epoch .....	57
3.151	Equivalence Class.....	57
3.152	Era.....	57
3.153	Event Management.....	57
3.154	Executable File.....	57
3.155	Execute.....	58
3.156	Execution Time .....	58
3.157	Execution Time Monitoring.....	58
3.158	Expand.....	58
3.159	Extended Regular Expression (ERE).....	58
3.160	Extended Security Controls.....	58
3.161	Feature Test Macro .....	59
3.162	Field.....	59
3.163	FIFO Special File (or FIFO) .....	59
3.164	File.....	59
3.165	File Description .....	59
3.166	File Descriptor .....	60
3.167	File Group Class .....	60
3.168	File Mode.....	60

Contents

3.169	File Mode Bits .....	60
3.170	Filename .....	60
3.171	File Offset .....	60
3.172	File Other Class .....	61
3.173	File Owner Class .....	61
3.174	File Permission Bits .....	61
3.175	File Serial Number .....	61
3.176	File System .....	61
3.177	File Type .....	61
3.178	Filter .....	62
3.179	First Open (of a File) .....	62
3.180	Flow Control .....	62
3.181	Foreground Job .....	62
3.182	Foreground Process .....	62
3.183	Foreground Process Group (or Foreground Job) .....	62
3.184	Foreground Process Group ID .....	62
3.185	Form-Feed Character (<form-feed>) .....	63
3.186	Graphic Character .....	63
3.187	Group Database .....	63
3.188	Group ID .....	63
3.189	Group Name .....	63
3.190	Hard Limit .....	63
3.191	Hard Link .....	64
3.192	Home Directory .....	64
3.193	Host Byte Order .....	64
3.194	Incomplete Line .....	64
3.195	Inf .....	64
3.196	Instrumented Application .....	64
3.197	Interactive Shell .....	64
3.198	Internationalization .....	65
3.199	Interprocess Communication .....	65
3.200	Invoke .....	65
3.201	Job .....	65
3.202	Job Control .....	65
3.203	Job Control Job ID .....	65
3.204	Last Close (of a File) .....	66
3.205	Line .....	66
3.206	Linger .....	66
3.207	Link .....	66
3.208	Link Count .....	66
3.209	Local Customs .....	66
3.210	Local Interprocess Communication (Local IPC) .....	66
3.211	Locale .....	67
3.212	Localization .....	67
3.213	Login .....	67
3.214	Login Name .....	67
3.215	Map .....	67
3.216	Marked Message .....	67
3.217	Matched .....	68
3.218	Memory Mapped Files .....	68
3.219	Memory Object .....	68
3.220	Memory-Resident .....	68

## Contents

3.221	Message .....	68
3.222	Message Catalog.....	68
3.223	Message Catalog Descriptor .....	69
3.224	Message Queue.....	69
3.225	Mode .....	69
3.226	Monotonic Clock .....	69
3.227	Mount Point .....	69
3.228	Multi-Character Collating Element .....	69
3.229	Mutex .....	69
3.230	Name .....	70
3.231	Named STREAM.....	70
3.232	NaN (Not a Number) .....	70
3.233	Native Language .....	70
3.234	Negative Response.....	70
3.235	Network.....	70
3.236	Network Address.....	70
3.237	Network Byte Order .....	71
3.238	Newline Character (<newline>) .....	71
3.239	Nice Value .....	71
3.240	Non-Blocking.....	71
3.241	Non-Spacing Characters .....	71
3.242	NUL.....	72
3.243	Null Byte.....	72
3.244	Null Pointer.....	72
3.245	Null String.....	72
3.246	Null Wide-Character Code .....	72
3.247	Number-Sign Character (<number-sign>) .....	72
3.248	Object File .....	72
3.249	Octet .....	72
3.250	Offset Maximum .....	73
3.251	Opaque Address.....	73
3.252	Open File .....	73
3.253	Open File Description.....	73
3.254	Operand.....	73
3.255	Operator .....	73
3.256	Option .....	74
3.257	Option-Argument .....	74
3.258	Orientation .....	74
3.259	Orphaned Process Group.....	74
3.260	Page .....	74
3.261	Page Size .....	74
3.262	Parameter .....	75
3.263	Parent Directory .....	75
3.264	Parent Process.....	75
3.265	Parent Process ID .....	75
3.266	Pathname.....	75
3.267	Pathname Component.....	76
3.268	Path Prefix .....	76
3.269	Pattern.....	76
3.270	Period Character (<period>) .....	76
3.271	Permissions .....	76
3.272	Persistence.....	76

Contents

3.273	Pipe.....	77
3.274	Polling.....	77
3.275	Portable Character Set.....	77
3.276	Portable Filename Character Set.....	77
3.277	Positional Parameter.....	78
3.278	Preallocation .....	78
3.279	Preempted Process (or Thread).....	78
3.280	Previous Job .....	78
3.281	Printable Character .....	78
3.282	Printable File .....	78
3.283	Priority .....	79
3.284	Priority Band.....	79
3.285	Priority Inversion .....	79
3.286	Priority Scheduling.....	79
3.287	Priority-Based Scheduling.....	79
3.288	Privilege.....	79
3.289	Process .....	80
3.290	Process Group.....	80
3.291	Process Group ID .....	80
3.292	Process Group Leader.....	80
3.293	Process Group Lifetime .....	80
3.294	Process ID.....	81
3.295	Process Lifetime.....	81
3.296	Process Memory Locking.....	81
3.297	Process Termination.....	81
3.298	Process-To-Process Communication .....	81
3.299	Process Virtual Time .....	82
3.300	Program .....	82
3.301	Protocol.....	82
3.302	Pseudo-Terminal .....	82
3.303	Radix Character .....	82
3.304	Read-Only File System .....	82
3.305	Read-Write Lock.....	82
3.306	Real Group ID.....	83
3.307	Real Time .....	83
3.308	Realtime Signal Extension .....	83
3.309	Real User ID .....	83
3.310	Record .....	83
3.311	Redirection .....	83
3.312	Redirection Operator.....	84
3.313	Referenced Shared Memory Object .....	84
3.314	Refresh .....	84
3.315	Regular Expression .....	84
3.316	Region .....	84
3.317	Regular File .....	84
3.318	Relative Pathname .....	85
3.319	Relocatable File.....	85
3.320	Relocation.....	85
3.321	Requested Batch Service .....	85
3.322	(Time) Resolution .....	85
3.323	Robust Mutex.....	85
3.324	Root Directory .....	85

## Contents

3.325	Runnable Process (or Thread) .....	85
3.326	Running Process (or Thread).....	86
3.327	Saved Resource Limits .....	86
3.328	Saved Set-Group-ID .....	86
3.329	Saved Set-User-ID .....	86
3.330	Scheduling.....	86
3.331	Scheduling Allocation Domain .....	86
3.332	Scheduling Contention Scope .....	86
3.333	Scheduling Policy.....	87
3.334	Screen .....	87
3.335	Scroll.....	87
3.336	Semaphore.....	87
3.337	Session .....	88
3.338	Session Leader .....	88
3.339	Session Lifetime.....	88
3.340	Shared Memory Object.....	88
3.341	Shell .....	88
3.342	Shell, the .....	88
3.343	Shell Script.....	89
3.344	Signal.....	89
3.345	Signal Stack .....	89
3.346	Single-Quote Character .....	89
3.347	Slash Character (<slash>).....	89
3.348	Socket .....	89
3.349	Socket Address .....	89
3.350	Soft Limit .....	90
3.351	Source Code .....	90
3.352	Space Character (<space>).....	90
3.353	Spawn .....	90
3.354	Special Built-In.....	90
3.355	Special Parameter.....	91
3.356	Spin Lock.....	91
3.357	Sporadic Server.....	91
3.358	Standard Error .....	91
3.359	Standard Input.....	91
3.360	Standard Output .....	91
3.361	Standard Utilities .....	91
3.362	Stream .....	92
3.363	STREAM .....	92
3.364	STREAM End .....	92
3.365	STREAM Head .....	92
3.366	STREAMS Multiplexor.....	92
3.367	String.....	92
3.368	Subshell.....	93
3.369	Successfully Transferred .....	93
3.370	Supplementary Group ID .....	93
3.371	Suspended Job .....	93
3.372	Symbolic Constant .....	93
3.373	Symbolic Link .....	94
3.374	Synchronized Input and Output.....	94
3.375	Synchronized I/O Completion .....	94
3.376	Synchronized I/O Data Integrity Completion.....	94

Contents

3.377	Synchronized I/O File Integrity Completion .....	94
3.378	Synchronized I/O Operation .....	94
3.379	Synchronous I/O Operation.....	95
3.380	Synchronously-Generated Signal .....	95
3.381	System.....	95
3.382	System Boot.....	95
3.383	System Clock.....	95
3.384	System Console .....	95
3.385	System Crash .....	95
3.386	System Databases.....	96
3.387	System Documentation .....	96
3.388	System Process.....	96
3.389	System Reboot .....	96
3.390	System Trace Event .....	96
3.391	System-Wide .....	96
3.392	Tab Character (<tab>).....	97
3.393	Terminal (or Terminal Device) .....	97
3.394	Text Column.....	97
3.395	Text File.....	97
3.396	Thread .....	97
3.397	Thread ID .....	97
3.398	Thread List .....	98
3.399	Thread-Safe .....	98
3.400	Thread-Specific Data Key.....	98
3.401	Tilde Character (<tilde>).....	98
3.402	Timeouts .....	98
3.403	Timer .....	98
3.404	Timer Overrun.....	98
3.405	Token.....	99
3.406	Trace Analyzer Process.....	99
3.407	Trace Controller Process.....	99
3.408	Trace Event.....	99
3.409	Trace Event Type .....	99
3.410	Trace Event Type Mapping.....	99
3.411	Trace Filter.....	99
3.412	Trace Generation Version .....	99
3.413	Trace Log .....	100
3.414	Trace Point.....	100
3.415	Trace Stream .....	100
3.416	Trace Stream Identifier .....	100
3.417	Trace System .....	100
3.418	Traced Process.....	100
3.419	Tracing Status of a Trace Stream .....	100
3.420	Typed Memory Name Space .....	100
3.421	Typed Memory Object.....	101
3.422	Typed Memory Pool .....	101
3.423	Typed Memory Port.....	101
3.424	Unbind .....	101
3.425	Unit Data .....	101
3.426	Upshifting .....	101
3.427	User Database.....	101
3.428	User ID.....	102

3.429	User Name .....	102
3.430	User Trace Event.....	102
3.431	Utility .....	102
3.432	Variable.....	103
3.433	Vertical-Tab Character (<vertical-tab>).....	103
3.434	White Space.....	103
3.435	Wide-Character Code (C Language).....	103
3.436	Wide-Character Input/Output Functions.....	103
3.437	Wide-Character String.....	103
3.438	Word.....	104
3.439	Working Directory (or Current Working Directory).....	104
3.440	Worldwide Portability Interface .....	104
3.441	Write.....	104
3.442	XSI .....	104
3.443	XSI-Conformant .....	105
3.444	Zombie Process.....	105
3.445	±0 .....	105
<b>Chapter 4</b>	<b>General Concepts .....</b>	<b>107</b>
4.1	Concurrent Execution.....	107
4.2	Directory Protection.....	107
4.3	Extended Security Controls.....	107
4.4	File Access Permissions.....	108
4.5	File Hierarchy .....	108
4.6	Filenames.....	109
4.7	Filename Portability.....	109
4.8	File Times Update .....	109
4.9	Host and Network Byte Orders .....	110
4.10	Measurement of Execution Time .....	110
4.11	Memory Synchronization .....	110
4.12	Pathname Resolution.....	111
4.13	Process ID Reuse .....	112
4.14	Scheduling Policy.....	112
4.15	Seconds Since the Epoch .....	113
4.16	Semaphore.....	113
4.17	Thread-Safety.....	114
4.18	Tracing .....	114
4.19	Treatment of Error Conditions for Mathematical Functions .....	116
4.19.1	Domain Error .....	116
4.19.2	Pole Error.....	117
4.19.3	Range Error .....	117
4.20	Treatment of NaN Arguments for the Mathematical Functions .....	118
4.21	Utility .....	118
4.22	Variable Assignment.....	118
<b>Chapter 5</b>	<b>File Format Notation.....</b>	<b>121</b>
<b>Chapter 6</b>	<b>Character Set .....</b>	<b>125</b>
6.1	Portable Character Set .....	125
6.2	Character Encoding .....	128

Contents

	6.3	C Language Wide-Character Codes .....	129
	6.4	Character Set Description File.....	129
	6.4.1	State-Dependent Character Encodings.....	132
<b>Chapter</b>	<b>7</b>	<b>Locale .....</b>	<b>135</b>
	7.1	General.....	135
	7.2	POSIX Locale .....	136
	7.3	Locale Definition .....	136
	7.3.1	LC_CTYPE .....	139
	7.3.2	LC_COLLATE.....	146
	7.3.3	LC_MONETARY .....	154
	7.3.4	LC_NUMERIC.....	157
	7.3.5	LC_TIME .....	158
	7.3.6	LC_MESSAGES.....	164
	7.4	Locale Definition Grammar.....	165
	7.4.1	Locale Lexical Conventions.....	165
	7.4.2	Locale Grammar.....	166
<b>Chapter</b>	<b>8</b>	<b>Environment Variables .....</b>	<b>173</b>
	8.1	Environment Variable Definition.....	173
	8.2	Internationalization Variables .....	174
	8.3	Other Environment Variables.....	177
<b>Chapter</b>	<b>9</b>	<b>Regular Expressions .....</b>	<b>181</b>
	9.1	Regular Expression Definitions.....	181
	9.2	Regular Expression General Requirements.....	182
	9.3	Basic Regular Expressions .....	183
	9.3.1	BREs Matching a Single Character or Collating Element .....	183
	9.3.2	BRE Ordinary Characters.....	183
	9.3.3	BRE Special Characters .....	183
	9.3.4	Periods in BREs .....	184
	9.3.5	RE Bracket Expression.....	184
	9.3.6	BREs Matching Multiple Characters .....	186
	9.3.7	BRE Precedence .....	187
	9.3.8	BRE Expression Anchoring.....	187
	9.4	Extended Regular Expressions.....	188
	9.4.1	EREs Matching a Single Character or Collating Element .....	188
	9.4.2	ERE Ordinary Characters.....	188
	9.4.3	ERE Special Characters .....	188
	9.4.4	Periods in EREs .....	189
	9.4.5	ERE Bracket Expression .....	189
	9.4.6	EREs Matching Multiple Characters .....	189
	9.4.7	ERE Alternation.....	190
	9.4.8	ERE Precedence .....	190
	9.4.9	ERE Expression Anchoring.....	190
	9.5	Regular Expression Grammar .....	191
	9.5.1	BRE/ERE Grammar Lexical Conventions.....	191
	9.5.2	RE and Bracket Expression Grammar.....	192
	9.5.3	ERE Grammar.....	194

<b>Chapter</b>	<b>10</b>	<b>Directory Structure and Devices .....</b>	<b>197</b>
	10.1	Directory Structure and Files.....	197
	10.2	Output Devices and Terminal Types.....	198
<b>Chapter</b>	<b>11</b>	<b>General Terminal Interface .....</b>	<b>199</b>
	11.1	Interface Characteristics .....	199
	11.1.1	Opening a Terminal Device File.....	199
	11.1.2	Process Groups .....	200
	11.1.3	The Controlling Terminal.....	200
	11.1.4	Terminal Access Control .....	201
	11.1.5	Input Processing and Reading Data .....	201
	11.1.6	Canonical Mode Input Processing.....	202
	11.1.7	Non-Canonical Mode Input Processing.....	202
	11.1.8	Writing Data and Output Processing .....	203
	11.1.9	Special Characters .....	203
	11.1.10	Modem Disconnect .....	205
	11.1.11	Closing a Terminal Device File.....	205
	11.2	Parameters that Can be Set .....	205
	11.2.1	The termios Structure .....	205
	11.2.2	Input Modes.....	206
	11.2.3	Output Modes.....	207
	11.2.4	Control Modes .....	209
	11.2.5	Local Modes .....	210
	11.2.6	Special Control Characters.....	212
<b>Chapter</b>	<b>12</b>	<b>Utility Conventions.....</b>	<b>213</b>
	12.1	Utility Argument Syntax.....	213
	12.2	Utility Syntax Guidelines.....	215
<b>Chapter</b>	<b>13</b>	<b>Headers .....</b>	<b>219</b>
<b>Volume</b>	<b>2</b>	<b>System Interfaces, Issue 7.....</b>	<b>463</b>
<b>Chapter</b>	<b>1</b>	<b>Introduction.....</b>	<b>465</b>
	1.1	Relationship to Other Formal Standards .....	465
	1.2	Format of Entries .....	465
<b>Chapter</b>	<b>2</b>	<b>General Information .....</b>	<b>467</b>
	2.1	Use and Implementation of Interfaces .....	467
	2.1.1	Use and Implementation of Functions.....	467
	2.1.2	Use and Implementation of Macros .....	468
	2.2	The Compilation Environment .....	468
	2.2.1	POSIX.1 Symbols.....	468
	2.2.2	The Name Space.....	469
	2.3	Error Numbers.....	477
	2.3.1	Additional Error Numbers .....	484
	2.4	Signal Concepts .....	484
	2.4.1	Signal Generation and Delivery.....	484
	2.4.2	Realtime Signal Generation and Delivery .....	485
	2.4.3	Signal Actions .....	486
	2.4.4	Signal Effects on Other Functions.....	490

Contents

2.5	Standard I/O Streams .....	490
2.5.1	Interaction of File Descriptors and Standard I/O Streams .....	491
2.5.2	Stream Orientation and Encoding Rules .....	493
2.6	STREAMS .....	494
2.6.1	Accessing STREAMS .....	495
2.7	XSI Interprocess Communication .....	496
2.7.1	IPC General Description .....	496
2.8	Realtime .....	497
2.8.1	Realtime Signals .....	497
2.8.2	Asynchronous I/O .....	497
2.8.3	Memory Management .....	499
2.8.4	Process Scheduling .....	501
2.8.5	Clocks and Timers .....	505
2.9	Threads .....	507
2.9.1	Thread-Safety .....	507
2.9.2	Thread IDs .....	508
2.9.3	Thread Mutexes .....	508
2.9.4	Thread Scheduling .....	509
2.9.5	Thread Cancellation .....	511
2.9.6	Thread Read-Write Locks .....	515
2.9.7	Thread Interactions with Regular File Operations .....	516
2.9.8	Use of Application-Managed Thread Stacks .....	516
2.10	Sockets .....	517
2.10.1	Address Families .....	517
2.10.2	Addressing .....	517
2.10.3	Protocols .....	517
2.10.4	Routing .....	518
2.10.5	Interfaces .....	518
2.10.6	Socket Types .....	518
2.10.7	Socket I/O Mode .....	519
2.10.8	Socket Owner .....	519
2.10.9	Socket Queue Limits .....	519
2.10.10	Pending Error .....	519
2.10.11	Socket Receive Queue .....	520
2.10.12	Socket Out-of-Band Data State .....	520
2.10.13	Connection Indication Queue .....	521
2.10.14	Signals .....	521
2.10.15	Asynchronous Errors .....	521
2.10.16	Use of Options .....	522
2.10.17	Use of Sockets for Local UNIX Connections .....	525
2.10.18	Use of Sockets over Internet Protocols .....	525
2.10.19	Use of Sockets over Internet Protocols Based on IPv4 .....	526
2.10.20	Use of Sockets over Internet Protocols Based on IPv6 .....	526
2.11	Tracing .....	529
2.11.1	Tracing Data Definitions .....	531
2.11.2	Trace Event Type Definitions .....	535
2.11.3	Trace Functions .....	539
2.12	Data Types .....	540
2.12.1	Defined Types .....	540

	2.12.2	The char Type.....	541
	2.12.3	Pointer Types .....	541
<b>Chapter</b>	<b>3</b>	<b>System Interfaces.....</b>	<b>543</b>
<b>Volume</b>	<b>3</b>	<b>Shell and Utilities, Issue 7 .....</b>	<b>2277</b>
<b>Chapter</b>	<b>1</b>	<b>Introduction.....</b>	<b>2279</b>
	1.1	Relationship to Other Documents .....	2279
	1.1.1	System Interfaces.....	2279
	1.1.2	Concepts Derived from the ISO C Standard .....	2283
	1.2	Utility Limits.....	2285
	1.3	Grammar Conventions.....	2287
	1.4	Utility Description Defaults.....	2288
	1.5	Considerations for Utilities in Support of Files of Arbitrary Size.....	2295
	1.6	Built-In Utilities .....	2296
<b>Chapter</b>	<b>2</b>	<b>Shell Command Language .....</b>	<b>2297</b>
	2.1	Shell Introduction.....	2297
	2.2	Quoting.....	2298
	2.2.1	Escape Character (Backslash).....	2298
	2.2.2	Single-Quotes.....	2298
	2.2.3	Double-Quotes.....	2298
	2.3	Token Recognition.....	2299
	2.3.1	Alias Substitution.....	2300
	2.4	Reserved Words.....	2301
	2.5	Parameters and Variables.....	2301
	2.5.1	Positional Parameters .....	2301
	2.5.2	Special Parameters .....	2302
	2.5.3	Shell Variables.....	2302
	2.6	Word Expansions .....	2305
	2.6.1	Tilde Expansion.....	2305
	2.6.2	Parameter Expansion.....	2306
	2.6.3	Command Substitution .....	2309
	2.6.4	Arithmetic Expansion.....	2310
	2.6.5	Field Splitting .....	2311
	2.6.6	Pathname Expansion .....	2311
	2.6.7	Quote Removal.....	2311
	2.7	Redirection .....	2312
	2.7.1	Redirecting Input .....	2312
	2.7.2	Redirecting Output.....	2313
	2.7.3	Appending Redirected Output .....	2313
	2.7.4	Here-Document.....	2313
	2.7.5	Duplicating an Input File Descriptor .....	2314
	2.7.6	Duplicating an Output File Descriptor .....	2314
	2.7.7	Open File Descriptors for Reading and Writing.....	2315
	2.8	Exit Status and Errors.....	2315
	2.8.1	Consequences of Shell Errors .....	2315
	2.8.2	Exit Status for Commands .....	2315
	2.9	Shell Commands .....	2316

Contents

2.9.1	Simple Commands.....	2316
2.9.2	Pipelines .....	2318
2.9.3	Lists .....	2319
2.9.4	Compound Commands.....	2321
2.9.5	Function Definition Command .....	2324
2.10	Shell Grammar.....	2325
2.10.1	Shell Grammar Lexical Conventions.....	2325
2.10.2	Shell Grammar Rules.....	2325
2.11	Signals and Error Handling.....	2330
2.12	Shell Execution Environment.....	2331
2.13	Pattern Matching Notation .....	2332
2.13.1	Patterns Matching a Single Character.....	2332
2.13.2	Patterns Matching Multiple Characters.....	2332
2.13.3	Patterns Used for Filename Expansion.....	2333
2.14	Special Built-In Utilities.....	2334
<b>Chapter 3</b>	<b>Batch Environment Services.....</b>	<b>2375</b>
3.1	General Concepts .....	2375
3.1.1	Batch Client-Server Interaction .....	2375
3.1.2	Batch Queues .....	2376
3.1.3	Batch Job Creation.....	2376
3.1.4	Batch Job Tracking.....	2376
3.1.5	Batch Job Routing.....	2377
3.1.6	Batch Job Execution .....	2377
3.1.7	Batch Job Exit.....	2378
3.1.8	Batch Job Abort.....	2378
3.1.9	Batch Authorization.....	2378
3.1.10	Batch Administration .....	2378
3.1.11	Batch Notification.....	2379
3.2	Batch Services .....	2379
3.2.1	Batch Job States.....	2380
3.2.2	Deferred Batch Services.....	2381
3.2.3	Requested Batch Services.....	2390
3.3	Common Behavior for Batch Environment Utilities.....	2397
3.3.1	Batch Job Identifier.....	2397
3.3.2	Destination .....	2398
3.3.3	Multiple Keyword-Value Pairs.....	2399
<b>Chapter 4</b>	<b>Utilities.....</b>	<b>2401</b>
<b>Volume 4</b>	<b>Rationale (Informative), Issue 7.....</b>	<b>3407</b>
<b>Part A</b>	<b>Base Definitions .....</b>	<b>3409</b>
<b>Appendix A</b>	<b>Rationale for Base Definitions.....</b>	<b>3411</b>
A.1	Introduction .....	3411
A.1.1	Scope .....	3411
A.1.2	Conformance.....	3414
A.1.3	Normative References .....	3414
A.1.4	Change History .....	3414
A.1.5	Terminology .....	3414

## Contents

A.1.6	Definitions and Concepts.....	3416
A.1.7	Portability.....	3416
A.2	Conformance.....	3417
A.2.1	Implementation Conformance.....	3417
A.2.2	Application Conformance.....	3421
A.2.3	Language-Dependent Services for the C Programming Language.....	3421
A.2.4	Other Language-Related Specifications.....	3422
A.3	Definitions.....	3422
A.4	General Concepts.....	3443
A.4.1	Concurrent Execution.....	3443
A.4.2	Directory Protection.....	3444
A.4.3	Extended Security Controls.....	3444
A.4.4	File Access Permissions.....	3444
A.4.5	File Hierarchy.....	3444
A.4.6	Filenames.....	3445
A.4.7	Filename Portability.....	3446
A.4.8	File Times Update.....	3446
A.4.9	Host and Network Byte Order.....	3447
A.4.10	Measurement of Execution Time.....	3447
A.4.11	Memory Synchronization.....	3447
A.4.12	Pathname Resolution.....	3449
A.4.13	Process ID Reuse.....	3450
A.4.14	Scheduling Policy.....	3450
A.4.15	Seconds Since the Epoch.....	3450
A.4.16	Semaphore.....	3452
A.4.17	Thread-Safety.....	3452
A.4.18	Tracing.....	3452
A.4.19	Treatment of Error Conditions for Mathematical Functions.....	3452
A.4.20	Treatment of NaN Arguments for Mathematical Functions.....	3452
A.4.21	Utility.....	3452
A.4.22	Variable Assignment.....	3452
A.5	File Format Notation.....	3452
A.6	Character Set.....	3453
A.6.1	Portable Character Set.....	3453
A.6.2	Character Encoding.....	3454
A.6.3	C Language Wide-Character Codes.....	3454
A.6.4	Character Set Description File.....	3454
A.7	Locale.....	3456
A.7.1	General.....	3456
A.7.2	POSIX Locale.....	3457
A.7.3	Locale Definition.....	3457
A.7.4	Locale Definition Grammar.....	3464
A.7.5	Locale Definition Example.....	3464
A.8	Environment Variables.....	3467
A.8.1	Environment Variable Definition.....	3467
A.8.2	Internationalization Variables.....	3468
A.8.3	Other Environment Variables.....	3469
A.9	Regular Expressions.....	3470
A.9.1	Regular Expression Definitions.....	3471

Contents

A.9.2	Regular Expression General Requirements.....	3471
A.9.3	Basic Regular Expressions .....	3472
A.9.4	Extended Regular Expressions.....	3475
A.9.5	Regular Expression Grammar.....	3477
A.10	Directory Structure and Devices.....	3478
A.10.1	Directory Structure and Files.....	3478
A.10.2	Output Devices and Terminal Types.....	3478
A.11	General Terminal Interface .....	3478
A.11.1	Interface Characteristics.....	3479
A.11.2	Parameters that Can be Set.....	3483
A.12	Utility Conventions.....	3485
A.12.1	Utility Argument Syntax.....	3485
A.12.2	Utility Syntax Guidelines.....	3486
A.13	Headers.....	3488
A.13.1	Format of Entries.....	3488
A.13.2	Removed Headers in Issue 7.....	3489
<b>Part B</b>	<b>System Interfaces.....</b>	<b>3491</b>
<b>Appendix B</b>	<b>Rationale for System Interfaces.....</b>	<b>3493</b>
B.1	Introduction .....	3493
B.1.1	Change History .....	3493
B.1.2	Relationship to Other Formal Standards.....	3496
B.1.3	Format of Entries.....	3496
B.2	General Information.....	3497
B.2.1	Use and Implementation of Interfaces.....	3497
B.2.2	The Compilation Environment .....	3498
B.2.3	Error Numbers.....	3503
B.2.4	Signal Concepts .....	3507
B.2.5	Standard I/O Streams .....	3517
B.2.6	STREAMS.....	3517
B.2.7	XSI Interprocess Communication .....	3518
B.2.8	Realtime.....	3519
B.2.9	Threads .....	3564
B.2.10	Sockets .....	3592
B.2.11	Tracing .....	3594
B.2.12	Data Types.....	3620
B.3	System Interfaces.....	3622
B.3.1	System Interfaces Removed in this Version .....	3622
B.3.2	System Interfaces Removed in the Previous Version.....	3625
B.3.3	Examples for Spawn.....	3625
<b>Part C</b>	<b>Shell and Utilities .....</b>	<b>3635</b>
<b>Appendix C</b>	<b>Rationale for Shell and Utilities.....</b>	<b>3637</b>
C.1	Introduction .....	3637
C.1.1	Change History .....	3637
C.1.2	Relationship to Other Documents .....	3638
C.1.3	Utility Limits.....	3639
C.1.4	Grammar Conventions.....	3642
C.1.5	Utility Description Defaults.....	3642

C.1.6	Considerations for Utilities in Support of Files of Arbitrary Size .....	3645
C.1.7	Built-In Utilities .....	3646
C.2	Shell Command Language .....	3648
C.2.1	Shell Introduction .....	3648
C.2.2	Quoting .....	3648
C.2.3	Token Recognition .....	3650
C.2.4	Reserved Words .....	3651
C.2.5	Parameters and Variables .....	3651
C.2.6	Word Expansions .....	3654
C.2.7	Redirection .....	3660
C.2.8	Exit Status and Errors .....	3662
C.2.9	Shell Commands .....	3662
C.2.10	Shell Grammar .....	3669
C.2.11	Signals and Error Handling .....	3671
C.2.12	Shell Execution Environment .....	3671
C.2.13	Pattern Matching Notation .....	3671
C.2.14	Special Built-In Utilities .....	3673
C.3	Batch Environment Services and Utilities .....	3673
C.3.1	Batch General Concepts .....	3676
C.3.2	Batch Services .....	3678
C.3.3	Common Behavior for Batch Environment Utilities .....	3679
C.4	Utilities .....	3679
C.4.1	Utilities Removed in this Version .....	3679
C.4.2	Utilities Removed in the Previous Version .....	3679
C.4.3	Exclusion of Utilities .....	3679
<b>Part</b>	<b>D Portability Considerations .....</b>	<b>3683</b>
<b>Appendix</b>	<b>D Portability Considerations (Informative) .....</b>	<b>3685</b>
D.1	User Requirements .....	3685
D.1.1	Configuration Interrogation .....	3686
D.1.2	Process Management .....	3686
D.1.3	Access to Data .....	3686
D.1.4	Access to the Environment .....	3686
D.1.5	Access to Determinism and Performance Enhancements .....	3686
D.1.6	Operating System-Dependent Profile .....	3687
D.1.7	I/O Interaction .....	3687
D.1.8	Internationalization Interaction .....	3687
D.1.9	C-Language Extensions .....	3687
D.1.10	Command Language .....	3687
D.1.11	Interactive Facilities .....	3687
D.1.12	Accomplish Multiple Tasks Simultaneously .....	3687
D.1.13	Complex Data Manipulation .....	3688
D.1.14	File Hierarchy Manipulation .....	3688
D.1.15	Locale Configuration .....	3688
D.1.16	Inter-User Communication .....	3688
D.1.17	System Environment .....	3688
D.1.18	Printing .....	3688
D.1.19	Software Development .....	3688

Contents

D.2	Portability Capabilities.....	3689
D.2.1	Configuration Interrogation .....	3689
D.2.2	Process Management .....	3690
D.2.3	Access to Data.....	3690
D.2.4	Access to the Environment .....	3691
D.2.5	Bounded (Realtime) Response .....	3692
D.2.6	Operating System-Dependent Profile .....	3692
D.2.7	I/O Interaction .....	3692
D.2.8	Internationalization Interaction .....	3693
D.2.9	C-Language Extensions.....	3693
D.2.10	Command Language .....	3693
D.2.11	Interactive Facilities .....	3694
D.2.12	Accomplish Multiple Tasks Simultaneously .....	3694
D.2.13	Complex Data Manipulation .....	3694
D.2.14	File Hierarchy Manipulation .....	3695
D.2.15	Locale Configuration .....	3695
D.2.16	Inter-User Communication.....	3695
D.2.17	System Environment .....	3696
D.2.18	Printing .....	3696
D.2.19	Software Development.....	3696
D.2.20	Future Growth .....	3696
D.3	Profiling Considerations .....	3697
D.3.1	Configuration Options .....	3697
D.3.2	Configuration Options (Shell and Utilities) .....	3697
D.3.3	Configurable Limits .....	3699
D.3.4	Configuration Options (System Interfaces).....	3699
D.3.5	Configurable Limits .....	3704
D.3.6	Optional Behavior .....	3707
<b>Part E</b>	<b>Subprofiling Considerations.....</b>	<b>3709</b>
<b>Appendix E</b>	<b>Subprofiling Considerations (Informative) .....</b>	<b>3711</b>
E.1	Subprofiling Option Groups.....	3711
	<b>Index .....</b>	<b>3717</b>
<b>List of Figures</b>		
B-1	Example of a System with Typed Memory .....	3537
B-2	Trace System Overview: for Offline Analysis .....	3600
B-3	Trace System Overview: for Online Analysis .....	3601
B-4	Trace System Overview: States of a Trace Stream .....	3603
B-5	Trace Another Process .....	3613
B-6	Trace Name Space Overview: With Third-Party Library .....	3614
<b>List of Tables</b>		
3-1	Job Control Job ID Formats.....	66
5-1	Escape Sequences and Associated Actions .....	121
6-1	Portable Character Set .....	125
6-2	Control Character Set .....	130

## Contents

7-1	Valid Character Class Combinations.....	142
10-1	Control Character Names .....	198
2-1	Value of Level for Socket Options.....	522
2-2	Socket-Level Options .....	523
2-3	Trace Option: System Trace Events.....	537
2-4	Trace and Trace Event Filter Options: System Trace Events.....	537
2-5	Trace and Trace Log Options: System Trace Events .....	538
2-6	Trace, Trace Log, and Trace Event Filter Options: System Trace Events .....	538
2-7	Trace Option: User Trace Event.....	539
1-1	Actions when Creating a File that Already Exists.....	2281
1-2	Selected ISO C Standard Operators and Control Flow Keywords .....	2284
1-3	Utility Limit Minimum Values.....	2285
1-4	Symbolic Utility Limits .....	2286
1-5	Regular Built-In Utilities .....	2296
3-1	Batch Utilities.....	2375
3-2	Environment Variable Summary .....	2379
3-3	Next State Table .....	2381
3-4	Results/Output Table .....	2383
3-5	Batch Services Summary .....	2390
A-1	Historical Practice for Symbolic Links.....	3440

## Trademarks

The following information is given for the convenience of users of POSIX.1-2008 and does not constitute an endorsement by the IEEE or The Open Group of these products. Equivalent products may be used if they can be shown to lead to the same results.

There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

754<sup>TM</sup>, 854<sup>TM</sup>, 1003.0<sup>TM</sup>, 1003.1<sup>TM</sup>, 1003.1d<sup>TM</sup>, 1003.1g<sup>TM</sup>, 1003.1j<sup>TM</sup>, 1003.1q<sup>TM</sup>, 1003.2<sup>TM</sup>, 1003.2a<sup>TM</sup>, 1003.2d<sup>TM</sup>, 1003.9<sup>TM</sup>, and 1003.13<sup>TM</sup> are trademarks of the Institute of Electrical and Electronic Engineers, Inc.

AIX<sup>®</sup> is a registered trademark of IBM Corporation.

AT&T<sup>®</sup> is a registered trademark of AT&T in the USA and other countries.

Boundaryless Information Flow<sup>TM</sup> and TOGAF<sup>TM</sup> are trademarks and Motif<sup>®</sup>, Making Standards Work<sup>®</sup>, OSF/1<sup>®</sup>, The Open Group<sup>®</sup>, UNIX<sup>®</sup>, and the "X" device are registered trademarks of The Open Group in the United States and other countries.

BSD<sup>TM</sup> is a trademark of the University of California, Berkeley, USA.

Hewlett-Packard<sup>®</sup>, HP<sup>®</sup>, and HP-UX<sup>®</sup> are registered trademarks of Hewlett-Packard Company.

IBM<sup>®</sup> is a registered trademark of International Business Machines Corporation.

IEEE<sup>®</sup> is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

POSIX<sup>®</sup> is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun<sup>®</sup> and Sun Microsystems<sup>®</sup> are registered trademarks of Sun Microsystems, Inc.

/usr/group<sup>®</sup> is a registered trademark of UniForum, the International Network of UNIX System Users.

## Acknowledgements

The contributions of the following organizations to the development of POSIX.1-2008 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems Inc. for permission to reproduce portions of their copyrighted documentation
- ISO/IEC JTC 1/SC 22/WG 14 C Language Committee
- Red Hat Inc. for permission to reproduce portions of its copyrighted documentation

POSIX.1-2008 was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Referenced Documents

### Normative References

Normative references for POSIX.1-2008 are defined in [Section 1.3](#) (on page 4).

### Informative References

The following documents are referenced in POSIX.1-2008:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Standards Terms

IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.

*Referenced Documents*

- IEEE Std 754<sup>TM</sup>-1985  
IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.
- IEEE Std 854<sup>TM</sup>-1987  
IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.
- IEEE Std 1003.9<sup>TM</sup>-1992  
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791  
Internet Protocol, Version 4 (IPv4), September 1981 (available at: [www.ietf.org/rfc/rfc0791.txt](http://www.ietf.org/rfc/rfc0791.txt)).
- IETF RFC 819  
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982 (available at: [www.ietf.org/rfc/rfc0819.txt](http://www.ietf.org/rfc/rfc0819.txt)).
- IETF RFC 822  
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982 (available at: [www.ietf.org/rfc/rfc0822.txt](http://www.ietf.org/rfc/rfc0822.txt)).
- IETF RFC 919  
Broadcasting Internet Datagrams, J. Mogul, October 1984 (available at: [www.ietf.org/rfc/rfc0919.txt](http://www.ietf.org/rfc/rfc0919.txt)).
- IETF RFC 920  
Domain Requirements, J. Postel, J. Reynolds, October 1984 (available at: [www.ietf.org/rfc/rfc0920.txt](http://www.ietf.org/rfc/rfc0920.txt)).
- IETF RFC 921  
Domain Name System Implementation Schedule, J. Postel, October 1984 (available at: [www.ietf.org/rfc/rfc0921.txt](http://www.ietf.org/rfc/rfc0921.txt)).
- IETF RFC 922  
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984 (available at: [www.ietf.org/rfc/rfc0922.txt](http://www.ietf.org/rfc/rfc0922.txt)).
- IETF RFC 1034  
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987 (available at: [www.ietf.org/rfc/rfc1034.txt](http://www.ietf.org/rfc/rfc1034.txt)).
- IETF RFC 1035  
Domain Names — Implementation and Specification, P. Mockapetris, November 1987 (available at: [www.ietf.org/rfc/rfc1035.txt](http://www.ietf.org/rfc/rfc1035.txt)).
- IETF RFC 1123  
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989 (available at: [www.ietf.org/rfc/rfc1123.txt](http://www.ietf.org/rfc/rfc1123.txt)).
- IETF RFC 1886  
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995 (available at: [www.ietf.org/rfc/rfc1886.txt](http://www.ietf.org/rfc/rfc1886.txt)).
- IETF RFC 2045  
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996 (available at: [www.ietf.org/rfc/rfc2045.txt](http://www.ietf.org/rfc/rfc2045.txt)).

*Referenced Documents*

- IETF RFC 2181  
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at: [www.ietf.org/rfc/rfc2181.txt](http://www.ietf.org/rfc/rfc2181.txt)).
- IETF RFC 2373  
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998 (available at: [www.ietf.org/rfc/rfc2373.txt](http://www.ietf.org/rfc/rfc2373.txt)).
- IETF RFC 2460  
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998 (available at: [www.ietf.org/rfc/rfc2460.txt](http://www.ietf.org/rfc/rfc2460.txt)).
- Internationalisation Guide  
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO 2375: 1985  
ISO 2375: 1985, Data Processing — Procedure for Registration of Escape Sequences.
- ISO 8652: 1987  
ISO 8652: 1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).
- ISO/IEC 1539: 1991  
ISO/IEC 1539: 1991, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).
- ISO/IEC 4873: 1991  
ISO/IEC 4873: 1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.
- ISO/IEC 6429: 1992  
ISO/IEC 6429: 1992, Information Technology — Control Functions for Coded Character Sets.
- ISO/IEC 6937: 1994  
ISO/IEC 6937: 1994, Information Technology — Coded Graphic Character Set for Text Communication — Latin Alphabet.
- ISO/IEC 8802-3: 1996  
ISO/IEC 8802-3: 1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.
- ISO/IEC 8859  
ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:  
Part 1: Latin Alphabet No. 1  
Part 2: Latin Alphabet No. 2  
Part 3: Latin Alphabet No. 3  
Part 4: Latin Alphabet No. 4  
Part 5: Latin/Cyrillic Alphabet  
Part 6: Latin/Arabic Alphabet  
Part 7: Latin/Greek Alphabet  
Part 8: Latin/Hebrew Alphabet  
Part 9: Latin Alphabet No. 5  
Part 10: Latin Alphabet No. 6  
Part 11: Latin/Thai Alphabet

*Referenced Documents*

- Part 13: Latin Alphabet No. 7
- Part 14: Latin Alphabet No. 8 (Celtic)
- Part 15: Latin Alphabet No. 9
- Part 16: Latin Alphabet No. 10

## ISO/IEC 9899: 1990

ISO/IEC 9899: 1990, Programming Languages — C, including Amendment 1: 1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

## ISO POSIX-1: 1996

ISO/IEC 9945-1: 1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

## ISO POSIX-2: 1993

ISO/IEC 9945-2: 1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2™-1992, as amended by ANSI/IEEE Std 1003.2a™-1992).

## Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

## Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

## Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

## Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

*Referenced Documents*

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Issue 6

Technical Standard, April 2004, published by The Open Group:

- Base Definitions (XBD), Issue 6 (ISBN: 1-931624-43-7, C046)
- System Interfaces (XSH), Issue 6 (ISBN: 1-931624-44-5, C047)
- Shell and Utilities (XCU), Issue 6 (ISBN: 1-931624-45-3, C048)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0™-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1™-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1™-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d™-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) —

## Referenced Documents

Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

POSIX.2d: 1994

IEEE Std 1003.2d™-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

POSIX.13: 1998

IEEE Std 1003.13™-1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

*Referenced Documents*

- XNS, Issue 4  
CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.
- XNS, Issue 5  
CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.
- XNS, Issue 5.2  
Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.
- X/Open Curses, Issue 4, Version 2  
CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.
- Yacc  
*Yacc: Yet Another Compiler Compiler*, Stephen C. Johnson, 1978.

**Source Documents**

Parts of the following documents were used to create the base documents for POSIX.1-2008:

- AIX 3.2 Manual  
AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).
- OSF/1  
OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).
- OSF AES  
Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).
- System V Release 2.0  
— UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).  
— UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).
- System V Release 4.2  
Operating System API Reference, UNIX<sup>®</sup> SVR4.2 (1992) (ISBN: 0-13-017658-3).

**Standard for Information Technology—  
Portable Operating System Interface (POSIX®)**

**Technical Standard: Base Specifications, Issue 7**

Prepared by the Austin Group ([www.opengroup.org/austin](http://www.opengroup.org/austin)).

IMPORTANT NOTICE: This standard is not intended to assure safety, security, health, or environmental protection in all circumstances. Implementors of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents”. They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IEEEDisclaimers.html>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**Technical Standard**

1 **Vol. 1:**  
2 **Base Definitions, Issue 7**

3 *The Open Group*  
4 *The Institute of Electrical and Electronics Engineers, Inc.*

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 1.1 Scope

POSIX.1-2008 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both application developers and system implementors.

POSIX.1-2008 comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of POSIX.1-2008, including utility conventions and C-language header definitions, are included in the Base Definitions volume of POSIX.1-2008.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of POSIX.1-2008.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of POSIX.1-2008.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of POSIX.1-2008 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of POSIX.1-2008.

The following areas are outside of the scope of POSIX.1-2008:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2008 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in POSIX.1-2008 are drawn from the following base documents:

- IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)

- 40 • The Open Group Technical Standard, 2006, Extended API Set Part 1
  - 41 • The Open Group Technical Standard, 2006, Extended API Set Part 2
  - 42 • The Open Group Technical Standard, 2006, Extended API Set Part 3
  - 43 • The Open Group Technical Standard, 2006, Extended API Set Part 4
  - 44 • ISO/IEC 9899:1999, Programming Languages — C, including ISO/IEC
  - 45 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC
  - 46 9899:1999/Cor.3
- 47 Emphasis has been placed on standardizing existing practice for existing users, with changes
- 48 and additions limited to correcting deficiencies in the following areas:
- 49 • Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,
  - 50 and ISO/IEC defect reports against ISO/IEC 9945
  - 51 • Issues raised in corrigenda for The Open Group Technical Standards and working group
  - 52 resolutions from The Open Group
  - 53 • Issues arising from ISO TR 24715:2006, Conflicts between POSIX and the LSB
  - 54 • Changes to make the text self-consistent with the additional material merged
  - 55 • Features, marked Legacy or obsolescent in the base documents, have been considered for
  - 56 removal in this version
  - 57 • A review and reorganization of the options within the standard
  - 58 • Alignment with the ISO/IEC 9899:1999 standard, including ISO/IEC
  - 59 9899:1999/Cor.2:2004(E)

## 60 1.2 Conformance

61 Conformance requirements for POSIX.1-2008 are defined in [Chapter 2](#) (on page 15).

## 62 1.3 Normative References

63 The following standards contain provisions which, through references in POSIX.1-2008,

64 constitute provisions of POSIX.1-2008. At the time of publication, the editions indicated were

65 valid. All standards are subject to revision, and parties to agreements based on POSIX.1-2008 are

66 encouraged to investigate the possibility of applying the most recent editions of the standards

67 listed below. Members of IEC and ISO maintain registers of currently valid International

68 Standards.

69 ANS X3.9-1978

70 (Reaffirmed 1989) American National Standard for Information Systems: Standard

71 X3.9-1978, Programming Language FORTRAN.<sup>1</sup>

72 ISO/IEC 646:1991

73 ISO/IEC 646:1991, Information Processing — ISO 7-Bit Coded Character Set for

74 Information Interchange.<sup>2</sup>

---

75 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY

76 10018, USA.

*Introduction**Normative References*

- 77 ISO 4217:2001  
78 ISO 4217:2001, Codes for the Representation of Currencies and Funds.
- 79 ISO 8601:2004  
80 ISO 8601:2004, Data Elements and Interchange Formats — Information Interchange —  
81 Representation of Dates and Times.
- 82 ISO C (1999)  
83 ISO/IEC 9899:1999, Programming Languages — C, including ISO/IEC  
84 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC  
85 9899:1999/Cor.3.
- 86 ISO/IEC 10646-1:2000  
87 ISO/IEC 10646-1:2000, Information Technology — Universal Multiple-Octet Coded  
88 Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

89 **1.4 Change History**

90 Change history is described in the Rationale (Informative) volume of POSIX.1-2008, and in the  
91 CHANGE HISTORY section of reference pages.

92 **1.5 Terminology**

93 For the purposes of POSIX.1-2008, the following terminology definitions apply:

94 **can**

95 Describes a permissible optional feature or behavior available to the user or application. The  
96 feature or behavior is mandatory for an implementation that conforms to POSIX.1-2008. An  
97 application can rely on the existence of the feature or behavior.

98 **implementation-defined**

99 Describes a value or behavior that is not defined by POSIX.1-2008 but is selected by an  
100 implementor. The value or behavior may vary among implementations that conform to  
101 POSIX.1-2008. An application should not rely on the existence of the value or behavior. An  
102 application that relies on such a value or behavior cannot be assured to be portable across  
103 conforming implementations.

104 The implementor shall document such a value or behavior so that it can be used correctly  
105 by an application.

106 **legacy**

107 Describes a feature or behavior that is being retained for compatibility with older  
108 applications, but which has limitations which make it inappropriate for developing portable  
109 applications. New applications should use alternative means of obtaining equivalent  
110 functionality.

111 **may**

112 Describes a feature or behavior that is optional for an implementation that conforms to  
113 POSIX.1-2008. An application should not rely on the existence of the feature or behavior. An  
114 application that relies on such a feature or behavior cannot be assured to be portable across  
115 conforming implementations.

116 2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse

117 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

118 **shall**

119 For an implementation that conforms to POSIX.1-2008, describes a feature or behavior that  
120 is mandatory. An application can rely on the existence of the feature or behavior.

121 For an application or user, describes a behavior that is mandatory.

122 **should**

123 For an implementation that conforms to POSIX.1-2008, describes a feature or behavior that  
124 is recommended but not mandatory. An application should not rely on the existence of the  
125 feature or behavior. An application that relies on such a feature or behavior cannot be  
126 assured to be portable across conforming implementations.

127 For an application, describes a feature or behavior that is recommended programming  
128 practice for optimum portability.

129 **undefined**

130 Describes the nature of a value or behavior not defined by POSIX.1-2008 which results from  
131 use of an invalid program construct or invalid data input.

132 The value or behavior may vary among implementations that conform to POSIX.1-2008. An  
133 application should not rely on the existence or validity of the value or behavior. An  
134 application that relies on any particular value or behavior cannot be assured to be portable  
135 across conforming implementations.

136 **unspecified**

137 Describes the nature of a value or behavior not specified by POSIX.1-2008 which results  
138 from use of a valid program construct or valid data input.

139 The value or behavior may vary among implementations that conform to POSIX.1-2008. An  
140 application should not rely on the existence or validity of the value or behavior. An  
141 application that relies on any particular value or behavior cannot be assured to be portable  
142 across conforming implementations.

143 **1.6 Definitions and Concepts**

144 Definitions and concepts are defined in [Chapter 3](#) (on page 33) and [Chapter 4](#) (on page 107).

145 **1.7 Portability**

146 Some of the utilities in the Shell and Utilities volume of POSIX.1-2008 and functions in the  
147 System Interfaces volume of POSIX.1-2008 describe functionality that might not be fully portable  
148 to systems meeting the requirements for POSIX conformance (see [Chapter 2](#), on page 15).

149 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
150 the margin identifies the nature of the option, extension, or warning (see [Section 1.7.1](#), on page  
151 7). For maximum portability, an application should avoid such functionality.

152 Unless the primary task of a utility is to produce textual material on its standard output,  
153 application developers should not rely on the format or content of any such material that may be  
154 produced. Where the primary task *is* to provide such material, but the output format is  
155 incompletely specified, the description is marked with the OF margin code and shading.  
156 Application developers are warned not to expect that the output of such an interface on one

157 system is any guide to its behavior on another system.

### 158 1.7.1 Codes

159 The codes and their meanings are as follows. See also [Section 1.7.2](#) (on page 13).

#### 160 ADV **Advisory Information**

161 The functionality described is optional. The functionality described is also an extension to the  
162 ISO C standard.

163 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
164 Where additional semantics apply to a function, the material is identified by use of the ADV  
165 margin legend.

#### 166 BE **Batch Environment Services and Utilities**

167 The functionality described is optional.

168 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.  
169 Where additional semantics apply to a utility, the material is identified by use of the BE margin  
170 legend.

#### 171 CD **C-Language Development Utilities**

172 The functionality described is optional.

173 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.  
174 Where additional semantics apply to a utility, the material is identified by use of the CD margin  
175 legend.

#### 176 CPT **Process CPU-Time Clocks**

177 The functionality described is optional. The functionality described is also an extension to the  
178 ISO C standard.

179 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.  
180 Where additional semantics apply to a function, the material is identified by use of the CPT  
181 margin legend.

#### 182 CX **Extension to the ISO C standard**

183 The functionality described is an extension to the ISO C standard. Application developers may  
184 make use of an extension as it is supported on all POSIX.1-2008-conforming systems.

185 With each function or header from the ISO C standard, a statement to the effect that “any  
186 conflict is unintentional” is included. That is intended to refer to a direct conflict. POSIX.1-2008  
187 acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors  
188 allowed to vary by the ISO C standard. Such limitations and other compatible differences are not  
189 considered conflicts, even if a CX mark is missing. The markings are for information only.

190 Where additional semantics apply to a function or header, the material is identified by use of the  
191 CX margin legend.

#### 192 FD **FORTTRAN Development Utilities**

193 The functionality described is optional.

194 Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.  
195 Where additional semantics apply to a utility, the material is identified by use of the FD margin  
196 legend.

#### 197 FR **FORTTRAN Runtime Utilities**

198 The functionality described is optional.

199 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.

200		Where additional semantics apply to a utility, the material is identified by use of the FR margin legend.
201		
202	FSC	<b>File Synchronization</b>
203		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
204		
205		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
206		Where additional semantics apply to a function, the material is identified by use of the FSC margin legend.
207		
208	IP6	<b>IPv6</b>
209		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
210		
211		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
212		Where additional semantics apply to a function, the material is identified by use of the IP6 margin legend.
213		
214	MC1	<b>Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust Mutex Priority Protection or Robust Mutex Priority Inheritance</b>
215		
216		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
217		
218		This is a shorthand notation for combinations of multiple option codes.
219		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
220		Where additional semantics apply to a function, the material is identified by use of the MC1 margin legend.
221		
222		Refer to <a href="#">Section 1.7.2</a> (on page 13).
223	ML	<b>Process Memory Locking</b>
224		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
225		
226		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
227		Where additional semantics apply to a function, the material is identified by use of the ML margin legend.
228		
229	MLR	<b>Range Memory Locking</b>
230		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
231		
232		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
233		Where additional semantics apply to a function, the material is identified by use of the MLR margin legend.
234		
235	MON	<b>Monotonic Clock</b>
236		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
237		
238		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
239		Where additional semantics apply to a function, the material is identified by use of the MON margin legend.
240		
241	MSG	<b>Message Passing</b>
242		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
243		
244		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.

- 245 Where additional semantics apply to a function, the material is identified by use of the MSG  
246 margin legend.
- 247 **MX** **IEC 60559 Floating-Point**  
248 The functionality described is optional. The functionality described is also an extension to the  
249 ISO C standard.
- 250 Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.  
251 Where additional semantics apply to a function, the material is identified by use of the MX  
252 margin legend.
- 253 **OB** **Obsolescent**  
254 The functionality described may be removed in a future version of this volume of POSIX.1-2008.  
255 Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use  
256 obsolescent features.
- 257 Where applicable, the material is identified by use of the OB margin legend.
- 258 **OF** **Output Format Incompletely Specified**  
259 The functionality described is an XSI extension. The format of the output produced by the  
260 utility is not fully specified. It is therefore not possible to post-process this output in a consistent  
261 fashion. Typical problems include unknown length of strings and unspecified field delimiters.
- 262 Where applicable, the material is identified by use of the OF margin legend.
- 263 **OH** **Optional Header**  
264 In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-2008 an  
265 included header is marked as in the following example:
- 266 **OH** `#include <sys/types.h>`  
267 `#include <grp.h>`  
268 `struct group *getgrnam(const char *name);`
- 269 The OH margin legend indicates that the marked header is not required on XSI-conformant  
270 systems.
- 271 **PIO** **Prioritized Input and Output**  
272 The functionality described is optional. The functionality described is also an extension to the  
273 ISO C standard.
- 274 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.  
275 Where additional semantics apply to a function, the material is identified by use of the PIO  
276 margin legend.
- 277 **PS** **Process Scheduling**  
278 The functionality described is optional. The functionality described is also an extension to the  
279 ISO C standard.
- 280 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.  
281 Where additional semantics apply to a function, the material is identified by use of the PS  
282 margin legend.
- 283 **RPI** **Robust Mutex Priority Inheritance**  
284 The functionality described is optional. The functionality described is also an extension to the  
285 ISO C standard.
- 286 Where applicable, functions are marked with the RPI margin legend in the SYNOPSIS section.  
287 Where additional semantics apply to a function, the material is identified by use of the RPI  
288 margin legend.

289	RPP	<b>Robust Mutex Priority Protection</b>
290		The functionality described is optional. The functionality described is also an extension to the
291		ISO C standard.
292		Where applicable, functions are marked with the RPP margin legend in the SYNOPSIS section.
293		Where additional semantics apply to a function, the material is identified by use of the RPP
294		margin legend.
295	RS	<b>Raw Sockets</b>
296		The functionality described is optional. The functionality described is also an extension to the
297		ISO C standard.
298		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
299		Where additional semantics apply to a function, the material is identified by use of the RS
300		margin legend.
301	SD	<b>Software Development Utilities</b>
302		The functionality described is optional.
303		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
304		Where additional semantics apply to a utility, the material is identified by use of the SD margin
305		legend.
306	SHM	<b>Shared Memory Objects</b>
307		The functionality described is optional. The functionality described is also an extension to the
308		ISO C standard.
309		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
310		Where additional semantics apply to a function, the material is identified by use of the SHM
311		margin legend.
312	SIO	<b>Synchronized Input and Output</b>
313		The functionality described is optional. The functionality described is also an extension to the
314		ISO C standard.
315		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
316		Where additional semantics apply to a function, the material is identified by use of the SIO
317		margin legend.
318	SPN	<b>Spawn</b>
319		The functionality described is optional. The functionality described is also an extension to the
320		ISO C standard.
321		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
322		Where additional semantics apply to a function, the material is identified by use of the SPN
323		margin legend.
324	SS	<b>Process Sporadic Server</b>
325		The functionality described is optional. The functionality described is also an extension to the
326		ISO C standard.
327		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
328		Where additional semantics apply to a function, the material is identified by use of the SS
329		margin legend.
330	TCT	<b>Thread CPU-Time Clocks</b>
331		The functionality described is optional. The functionality described is also an extension to the
332		ISO C standard.
333		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.

334		Where additional semantics apply to a function, the material is identified by use of the TCT margin legend.
335		
336	TEF	<b>Trace Event Filter</b>
337		The functionality described is optional. This functionality is dependent on support for the Trace option. The functionality described is also an extension to the ISO C standard.
338		
339		Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
340		Where additional semantics apply to a function, the material is identified by use of the TEF margin legend.
341		
342	TPI	<b>Non-Robust Mutex Priority Inheritance</b>
343		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
344		
345		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
346		Where additional semantics apply to a function, the material is identified by use of the TPI margin legend.
347		
348	TPP	<b>Non-Robust Mutex Priority Protection</b>
349		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
350		
351		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
352		Where additional semantics apply to a function, the material is identified by use of the TPP margin legend.
353		
354	TPS	<b>Thread Execution Scheduling</b>
355		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
356		
357		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
358		Where additional semantics apply to a function, the material is identified by use of the TPS margin legend.
359		
360	TRC	<b>Trace</b>
361		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
362		
363		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
364		Where additional semantics apply to a function, the material is identified by use of the TRC margin legend.
365		
366	TRI	<b>Trace Inherit</b>
367		The functionality described is optional. This functionality is dependent on support for the Trace option. The functionality described is also an extension to the ISO C standard.
368		
369		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
370		Where additional semantics apply to a function, the material is identified by use of the TRI margin legend.
371		
372	TRL	<b>Trace Log</b>
373		The functionality described is optional. This functionality is dependent on support for the Trace option. The functionality described is also an extension to the ISO C standard.
374		
375		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
376		Where additional semantics apply to a function, the material is identified by use of the TRL margin legend.
377		

378	TSA	<b>Thread Stack Address Attribute</b>
379		The functionality described is optional. The functionality described is also an extension to the
380		ISO C standard.
381		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
382		Where additional semantics apply to a function, the material is identified by use of the TSA
383		margin legend.
384	TSH	<b>Thread Process-Shared Synchronization</b>
385		The functionality described is optional. The functionality described is also an extension to the
386		ISO C standard.
387		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
388		Where additional semantics apply to a function, the material is identified by use of the TSH
389		margin legend.
390	TSP	<b>Thread Sporadic Server</b>
391		The functionality described is optional. The functionality described is also an extension to the
392		ISO C standard.
393		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
394		Where additional semantics apply to a function, the material is identified by use of the TSP
395		margin legend.
396	TSS	<b>Thread Stack Size Attribute</b>
397		The functionality described is optional. The functionality described is also an extension to the
398		ISO C standard.
399		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
400		Where additional semantics apply to a function, the material is identified by use of the TSS
401		margin legend.
402	TYM	<b>Typed Memory Objects</b>
403		The functionality described is optional. The functionality described is also an extension to the
404		ISO C standard.
405		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
406		Where additional semantics apply to a function, the material is identified by use of the TYM
407		margin legend.
408	UP	<b>User Portability Utilities</b>
409		The functionality described is optional.
410		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
411		Where additional semantics apply to a utility, the material is identified by use of the UP margin
412		legend.
413	UU	<b>UUCP Utilities</b>
414		The functionality described is optional. The functionality described is also an extension to the
415		ISO C standard.
416		Where applicable, functions are marked with the UU margin legend in the SYNOPSIS section.
417		Where additional semantics apply to a function, the material is identified by use of the UU
418		margin legend.
419	XSI	<b>X/Open System Interfaces</b>
420		The functionality described is part of the X/Open Systems Interfaces option. Functionality
421		marked XSI is an extension to the ISO C standard. Application developers may confidently
422		make use of such extensions on all systems supporting the X/Open System Interfaces option.

423 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that  
424 reference page is an extension. See [Section 2.1.4](#) (on page 19).

425 XSR **XSI STREAMS**  
426 The functionality described is optional. The functionality described is also an extension to the  
427 ISO C standard.

428 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.  
429 Where additional semantics apply to a function, the material is identified by use of the XSR  
430 margin legend.

## 431 1.7.2 Margin Code Notation

432 Some of the functionality described in POSIX.1-2008 depends on support of more than one  
433 option, or independently may depend on several options. The following notation for margin  
434 codes is used to denote the following cases.

### 435 A Feature Dependent on One or Two Options

436 In this case, margin codes have a <space> separator; for example:

437 SHM This feature requires support for only the Shared Memory Objects option.

438 SHM TYM This feature requires support for both the Shared Memory Objects option and the Typed  
439 Memory Objects option; that is, an application which uses this feature is portable only between  
440 implementations that provide both options.

### 441 A Feature Dependent on Either of the Options Denoted

442 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

443 SHM | TYM This feature is dependent on support for either the Shared Memory Objects option or the Typed  
444 Memory Objects option; that is, an application which uses this feature is portable between  
445 implementations that provide any (or all) of the options.

### 446 A Feature Dependent on More than Two Options

447 The following shorthand notations are used:

448 MC1 The MC1 margin code is shorthand for TPP | TPI | RPP | RPI. Features which are shaded with this  
449 margin code require support of either the Non-Robust Mutex Priority Protection option or the  
450 Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or  
451 the Robust Mutex Priority Inheritance option.

### 452 Large Sections Dependent on an Option

453 Where large sections of text are dependent on support for an option, a lead-in text block is  
454 provided and shaded accordingly; for example:

455 XSI This section describes extensions to support interprocess communication. The functionality  
456 described in this section shall be provided on implementations that support the XSI option (and  
457 the rest of this section is not further shaded).

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 2.1 Implementation Conformance

For the purposes of POSIX.1-2008, the implementation conformance requirements given in this section apply.

### 2.1.1 Requirements

A *conforming implementation* shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within POSIX.1-2008 that are required for POSIX conformance (see Section 2.1.3, on page 16). These interfaces shall support the functional behavior described herein.
2. The system may support the X/Open System Interfaces (XSI) option as described in Section 2.1.4 (on page 19).
3. The system may support one or more options as described under Section 2.1.5 (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
4. The system may provide non-standard extensions. These are features not required by POSIX.1-2008 and may include, but are not limited to:
  - Additional functions
  - Additional headers
  - Additional symbols in standard headers
  - Additional utilities
  - Additional options for standard utilities
  - Additional environment variables
  - Additional file types
  - Non-conforming file systems (for example, legacy file systems for which `_POSIX_NO_TRUNC` is false, case-insensitive file systems, or network file systems)
  - Dynamically populated file systems (for example, `/proc`)
  - Additional character special files with special properties (for example, `/dev/stdin`, `/dev/stdout`, and `/dev/stderr`)

Non-standard extensions of the utilities, functions, or facilities specified in POSIX.1-2008 should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by POSIX.1-2008. The conformance document shall define an environment in which an application can be run with the behavior specified by POSIX.1-2008. In no case shall such

492 an environment require modification of a Strictly Conforming POSIX Application (see  
493 [Section 2.2.1](#), on page 29).

## 494 2.1.2 Documentation

495 A conformance document with the following information shall be available for an  
496 implementation claiming conformance to POSIX.1-2008. The conformance document shall have  
497 the same structure as POSIX.1-2008, with the information presented in the appropriate sections  
498 and subsections. Sections and subsections that consist solely of subordinate section titles, with  
499 no other information, are not required. The conformance document shall not contain  
500 information about extended facilities or capabilities outside the scope of POSIX.1-2008.

501 The conformance document shall contain a statement that indicates the full name, number, and  
502 date of the standard that applies. The conformance document may also list international  
503 software standards that are available for use by a Conforming POSIX Application. Applicable  
504 characteristics where documentation is required by one of these standards, or by standards of  
505 government bodies, may also be included.

506 The conformance document shall describe the limit values found in the headers `<limits.h>` (on  
507 page 268) and `<unistd.h>` (on page 430), stating values, the conditions under which those values  
508 may change, and the limits of such variations, if any.

509 The conformance document shall describe the behavior of the implementation for all  
510 implementation-defined features defined in POSIX.1-2008. This requirement shall be met by  
511 listing these features and providing either a specific reference to the system documentation or  
512 providing full syntax and semantics of these features. When the value or behavior in the  
513 implementation is designed to be variable or customized on each instantiation of the system, the  
514 implementation provider shall document the nature and permissible ranges of this variation.

515 The conformance document may specify the behavior of the implementation for those features  
516 where POSIX.1-2008 states that implementations may vary or where features are identified as  
517 undefined or unspecified.

518 The conformance document shall not contain documentation other than that specified in the  
519 preceding paragraphs except where such documentation is specifically allowed or required by  
520 other provisions of POSIX.1-2008.

521 The phrases “shall document” or “shall be documented” in POSIX.1-2008 mean that  
522 documentation of the feature shall appear in the conformance document, as described  
523 previously, unless there is an explicit reference in the conformance document to show where the  
524 information can be found in the system documentation.

525 The system documentation should also contain the information found in the conformance  
526 document.

## 527 2.1.3 POSIX Conformance

528 A conforming implementation shall meet the following criteria for POSIX conformance.

## Conformance

## Implementation Conformance

## 529 2.1.3.1 POSIX System Interfaces

530 The following requirements apply to the system interfaces (functions and headers):

- 531 • The system shall support all the mandatory functions and headers defined in
- 532 POSIX.1-2008, and shall set the symbolic constant `_POSIX_VERSION` to the value 200809L.
- 533 • Although all implementations conforming to POSIX.1-2008 support all the features
- 534 described below, there may be system-dependent or file system-dependent configuration
- 535 procedures that can remove or modify any or all of these features. Such configurations
- 536 should not be made if strict compliance is required.

537 The following symbolic constants shall be defined with a value other than `-1`. If a constant

538 is defined with the value zero, applications should use the `sysconf()`, `pathconf()`, or

539 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the

540 system at that time or for the particular pathname in question.

541 — `_POSIX_CHOWN_RESTRICTED`

542 The use of `chown()` is restricted to a process with appropriate privileges, and to

543 changing the group ID of a file only to the effective group ID of the process or to one

544 of its supplementary group IDs.

545 — `_POSIX_NO_TRUNC`

546 Pathname components longer than `{NAME_MAX}` generate an error.

- 547 • The following symbolic constants shall be defined by the implementation as follows:

548 — Symbolic constants defined with the value 200809L:

549 `_POSIX_ASYNCHRONOUS_IO`

550 `_POSIX_BARRIERS`

551 `_POSIX_CLOCK_SELECTION`

552 `_POSIX_MAPPED_FILES`

553 `_POSIX_MEMORY_PROTECTION`

554 `_POSIX_READER_WRITER_LOCKS`

555 `_POSIX_REALTIME_SIGNALS`

556 `_POSIX_SEMAPHORES`

557 `_POSIX_SPIN_LOCKS`

558 `_POSIX_THREAD_SAFE_FUNCTIONS`

559 `_POSIX_THREADS`

560 `_POSIX_TIMEOUTS`

561 `_POSIX_TIMERS`

562 — Symbolic constants defined with a value greater than zero:

563 `_POSIX_JOB_CONTROL`

564 `_POSIX_SAVED_IDS`

565 — Symbolic constants defined with a value other than `-1`.

566 `_POSIX_VDISABLE`

567 **Note:** The symbols above represent historical options that are no longer allowed as options, but

568 are retained here for backwards-compatibility of applications.

- 569 • The system may support one or more options (see [Section 2.1.6](#), on page 26) denoted by the  
570 following symbolic constants:

571            \_POSIX\_ADVISORY\_INFO  
572            \_POSIX\_CPUTIME  
573            \_POSIX\_FSYNC  
574            \_POSIX\_IPV6  
575            \_POSIX\_MEMLOCK  
576            \_POSIX\_MEMLOCK\_RANGE  
577            \_POSIX\_MESSAGE\_PASSING  
578            \_POSIX\_MONOTONIC\_CLOCK  
579            \_POSIX\_PRIORITIZED\_IO  
580            \_POSIX\_PRIORITY\_SCHEDULING  
581            \_POSIX\_RAW\_SOCKETS  
582            \_POSIX\_SHARED\_MEMORY\_OBJECTS  
583            \_POSIX\_SPAWN  
584            \_POSIX\_SPORADIC\_SERVER  
585            \_POSIX\_SYNCHRONIZED\_IO  
586            \_POSIX\_THREAD\_ATTR\_STACKADDR  
587            \_POSIX\_THREAD\_CPUTIME  
588            \_POSIX\_THREAD\_ATTR\_STACKSIZE  
589            \_POSIX\_THREAD\_PRIO\_INHERIT  
590            \_POSIX\_THREAD\_PRIO\_PROTECT  
591            \_POSIX\_THREAD\_PRIORITY\_SCHEDULING  
592            \_POSIX\_THREAD\_PROCESS\_SHARED  
593            \_POSIX\_THREAD\_SPORADIC\_SERVER  
594            \_POSIX\_TRACE  
595            \_POSIX\_TRACE\_EVENT\_FILTER  
596            \_POSIX\_TRACE\_INHERIT  
597            \_POSIX\_TRACE\_LOG  
598            \_POSIX\_TYPED\_MEMORY\_OBJECTS  
599            \_XOPEN\_CRYPT  
600            \_XOPEN\_REALTIME  
601            \_XOPEN\_REALTIME\_THREADS  
602            \_XOPEN\_STREAMS  
603            \_XOPEN\_UNIX

604            If any of the symbolic constants `_POSIX_TRACE_EVENT_FILTER`, `_POSIX_TRACE_LOG`,  
605            or `_POSIX_TRACE_INHERIT` is defined to have a value other than `-1`, then the symbolic  
606            constant `_POSIX_TRACE` shall also be defined to have a value other than `-1`.

607            If the Advisory Information option is supported, there shall be at least one file system that  
608            supports the functionality.

### 609 2.1.3.2 POSIX Shell and Utilities

610            The following requirements apply to the shell and utilities:

- 611            • The system shall provide all the mandatory utilities in the Shell and Utilities volume of  
612            POSIX.1-2008 with all the functional behavior described therein.
- 613            • The system shall support the Large File capabilities described in the Shell and Utilities  
614            volume of POSIX.1-2008.

## Conformance

## Implementation Conformance

- 615 • The system may support one or more options (see [Section 2.1.6](#), on page 26) denoted by the  
616 following symbolic constants. (The literal names below apply to the *getconf* utility.)

617 POSIX2\_C\_DEV  
618 POSIX2\_CHAR\_TERM  
619 POSIX2\_FORT\_DEV  
620 POSIX2\_FORT\_RUN  
621 POSIX2\_LOCALEDEF  
622 POSIX2\_PBS  
623 POSIX2\_PBS\_ACCOUNTING  
624 POSIX2\_PBS\_LOCATE  
625 POSIX2\_PBS\_MESSAGE  
626 POSIX2\_PBS\_TRACK  
627 POSIX2\_SW\_DEV  
628 POSIX2\_UPE  
629 XOPEN\_UNIX  
630 XOPEN\_UUCP

631 Additional language bindings and development utility options may be provided in other related  
632 standards or in a future version of this standard. In the former case, additional symbolic  
633 constants of the same general form as shown in this subsection should be defined by the related  
634 standard document and made available to the application without requiring POSIX.1-2008 to be  
635 updated.

## 636 2.1.4 XSI Conformance

637 XSI This section describes the criteria for implementations providing conformance to the X/Open  
638 System Interfaces (XSI) option (see [Section 3.442](#), on page 104). The functionality described in  
639 this section shall be provided on implementations that support the XSI option (and the rest of  
640 this section is not further shaded).

641 POSIX.1-2008 describes utilities, functions, and facilities offered to application programs by the  
642 X/Open System Interfaces (XSI) option. An XSI-conforming implementation shall meet the  
643 criteria for POSIX conformance and the following requirements listed in this section.

644 XSI-conforming implementations shall set the symbolic constant `_XOPEN_UNIX` to a value  
645 other than `-1` and shall set the symbolic constant `_XOPEN_VERSION` to the value 700.

### 646 2.1.4.1 XSI System Interfaces

647 The following requirements apply to the system interfaces when the XSI option is supported:

- 648 • The system shall support all the functions and headers defined in POSIX.1-2008 as part of  
649 the XSI option denoted by the XSI marking in the SYNOPSIS section, and any extensions  
650 marked with the XSI option marking (see [Section 1.7.1](#), on page 7) within the text.
- 651 • The system shall support the following options defined within POSIX.1-2008 (see [Section](#)  
652 [2.1.6](#), on page 26):

653 `_POSIX_FSYNC`  
654 `_POSIX_THREAD_ATTR_STACKADDR`  
655 `_POSIX_THREAD_ATTR_STACKSIZE`  
656 `_POSIX_THREAD_PROCESS_SHARED`

- 657 • The system may support the following XSI Option Groups (see [Section 2.1.5.2](#), on page 22)
- 658 defined within POSIX.1-2008:
- 659 — Encryption
- 660 — Realtime
- 661 — Advanced Realtime
- 662 — Realtime Threads
- 663 — Advanced Realtime Threads
- 664 — Tracing
- 665 — XSI STREAMS

666 2.1.4.2 XSI Shell and Utilities Conformance

667 The following requirements apply to the shell and utilities when the XSI option is supported:

- 668 • The system shall support all the utilities defined in the Shell and Utilities volume of
- 669 POSIX.1-2008 as part of the XSI option denoted by the XSI marking in the SYNOPSIS
- 670 section, and any extensions marked with the XSI option marking (see [Section 1.7.1](#), on
- 671 page 7) within the text.
- 672 • The system shall support the User Portability Utilities option and the Terminal
- 673 Characteristics option.
- 674 • The system shall support creation of locales (see [Chapter 7](#), on page 135).
- 675 • The C-language Development utility `c99` shall be supported.
- 676 • The XSI Development Utilities option may be supported. It consists of the following
- 677 software development utilities:

678	<i>admin</i>	<i>delta</i>	<i>rmdel</i>	<i>val</i>
679	<i>cflow</i>	<i>get</i>	<i>sact</i>	<i>what</i>
680	<i>ctags</i>	<i>nm</i>	<i>secs</i>	
681	<i>cxref</i>	<i>prs</i>	<i>unget</i>	

682 2.1.5 Option Groups

683 An Option Group is a group of related functions or options defined within the System Interfaces

684 volume of POSIX.1-2008.

685 If an implementation supports an Option Group, then the system shall support the functional

686 behavior described herein.

687 If an implementation does not support an Option Group, then the system need not support the

688 functional behavior described herein.

689 2.1.5.1 Subprofiling Considerations

690 Profiling standards supporting functional requirements less than that required in POSIX.1-2008

691 may subset both mandatory and optional functionality required for POSIX Conformance (see

692 [Section 2.1.3](#), on page 16) or XSI Conformance (see [Section 2.1.4](#), on page 19). Such profiles shall

693 organize the subsets into Subprofiling Option Groups.

694 XRAT Appendix E (on page 3711) describes a representative set of such Subprofiling Option  
 695 Groups for use by profiles applicable to specialized realtime systems. POSIX.1-2008 does not  
 696 require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols  
 697 defined in any header) or at runtime (via *sysconf()* or *getconf()*).

698 A Subprofiling Option Group may provide basic system functionality that other Subprofiling  
 699 Option Groups and other options depend upon.<sup>3</sup> If a profile of POSIX.1-2008 does not require an  
 700 implementation to provide a Subprofiling Option Group that provides features utilized by a  
 701 required Subprofiling Option Group (or option),<sup>4</sup> the profile shall specify<sup>5</sup> all of the following:

- 702 • Restricted or altered behavior of interfaces defined in POSIX.1-2008 that may differ on an  
 703 implementation of the profile
- 704 • Additional behaviors that may produce undefined or unspecified results
- 705 • Additional implementation-defined behavior that implementations shall be required to  
 706 document in the profile's conformance document

707 if any of the above is a result of the profile not requiring an interface required by POSIX.1-2008.

708 The following additional rules shall apply to all profiles of POSIX.1-2008:

- 709 • Any application that conforms to that profile shall also conform to POSIX.1-2008, unless  
 710 the application depends on the definition of a profile support indicator macro in  
 711 `<unistd.h>` (that is, a profile shall not require restricted, altered, or extended behaviors of  
 712 an implementation of POSIX.1-2008).
- 713 • Profiles are permitted to require the definition of a *profile support indicator macro* with a  
 714 name beginning `_POSIX_AEP_` in `<unistd.h>`.
- 715 • Profiles shall require the definition of the macro `_POSIX_SUBPROFILE` in `<unistd.h>` on  
 716 implementations that do not meet all of the requirements of a POSIX.1-conforming  
 717 implementation.
- 718 • Profiles are permitted to add additional requirements to the limits defined in `<limits.h>`  
 719 and `<stdint.h>`, subject to the following:

720 For the limits in `<limits.h>` and `<stdint.h>`:

- 721 — If the limit is specified as having a fixed value, it shall not be changed by a profile.
- 722 — If a limit is specified as having a minimum or maximum acceptable value, it may be  
 723 changed by a profile as follows:

724 A profile may increase a minimum acceptable value, but shall not make a  
 725 minimum acceptable value smaller.

---

726 3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are  
 727 needed by any interface in POSIX.1-2008 that parses a *path* argument. If a profile requires support for the Device Input and Output  
 728 profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname  
 729 resolution is to behave in that profile, how the `O_CREAT` flag to *open()* is to be handled (and the use of the character 'a' in the *mode*  
 730 argument of  *fopen()* when a filename argument names a file that does not exist), and specify lots of other details.

731 4. As an example, POSIX.1-2008 requires that implementations claiming to support the Range Memory Locking option also support the  
 732 Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that  
 733 the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor  
 734 would have to know to build an application or implementation conforming to the profile.

735 5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified  
 736 results.

- 737 — A profile may reduce a maximum acceptable value, but shall not make a  
738 maximum acceptable value larger.
- 739 • A profile shall not change a limit specified as having a minimum or maximum value into a  
740 limit specified as having a fixed value.
  - 741 • A profile shall not create new limits.
  - 742 • Any implementation that conforms to POSIX.1-2008 (including all options and extended  
743 limits required by the profile) shall also conform to that profile, except for the possible  
744 omission from `<unistd.h>` of a profile support indicator macro required by the profile.

#### 745 2.1.5.2 XSI Option Groups

746 XSI This section describes Option Groups to support the definition of XSI conformance within the  
747 System Interfaces volume of POSIX.1-2008. The functionality described in this section shall be  
748 provided on implementations that support the XSI option and the appropriate Option Group  
749 (and the rest of this section is not further shaded).

750 The following Option Groups are defined.

#### 751 Encryption

752 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes  
753 the following functions:

754 `crypt()`, `encrypt()`, `setkey()`

755 These functions are marked CRYPT.

756 Due to export restrictions on the decoding algorithm in some countries, implementations may  
757 be restricted in making these functions available. All the functions in the Encryption Option  
758 Group may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]` for the  
759 decryption operation.

760 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to  
761 a value other than `-1`.

#### 762 Realtime

763 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

764 This Option Group includes a set of realtime functions drawn from options within POSIX.1-2008  
765 (see Section 2.1.6, on page 26).

766 Where entire functions are included in the Option Group, the NAME section is marked with  
767 REALTIME. Where additional semantics have been added to existing pages, the new material is  
768 identified by use of the appropriate margin legend for the underlying option defined within  
769 POSIX.1-2008.

770 An implementation that claims conformance to this Option Group shall set  
771 `_XOPEN_REALTIME` to a value other than `-1`.

772 This Option Group consists of the set of the following options from within POSIX.1-2008 (see  
773 Section 2.1.6, on page 26):

## Conformance

## Implementation Conformance

774            \_POSIX\_FSYNC  
 775            \_POSIX\_MEMLOCK  
 776            \_POSIX\_MEMLOCK\_RANGE  
 777            \_POSIX\_MESSAGE\_PASSING  
 778            \_POSIX\_PRIORITIZED\_IO  
 779            \_POSIX\_PRIORITY\_SCHEDULING  
 780            \_POSIX\_SHARED\_MEMORY\_OBJECTS  
 781            \_POSIX\_SYNCHRONIZED\_IO

782            If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the  
 783            following symbolic constants shall be defined by the implementation to have the value 200809L:

784            \_POSIX\_MEMLOCK  
 785            \_POSIX\_MEMLOCK\_RANGE  
 786            \_POSIX\_MESSAGE\_PASSING  
 787            \_POSIX\_PRIORITY\_SCHEDULING  
 788            \_POSIX\_SHARED\_MEMORY\_OBJECTS  
 789            \_POSIX\_SYNCHRONIZED\_IO

790            The functionality associated with `_POSIX_FSYNC` shall always be supported on XSI-conformant  
 791            systems.

792            Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If  
 793            `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by  
 794            `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling  
 795            priority equal to a base scheduling priority minus `aiocbp->aio_reqprio`. If Thread Execution  
 796            Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 797            otherwise, the base scheduling priority is that of the calling thread. The implementation shall  
 798            also document for which files I/O prioritization is supported.

### 799            Advanced Realtime

800            An implementation that claims conformance to this Option Group shall also support the  
 801            Realtime Option Group.

802            Where entire functions are included in the Option Group, the NAME section is marked with  
 803            ADVANCED REALTIME. Where additional semantics have been added to existing pages, the  
 804            new material is identified by use of the appropriate margin legend for the underlying option  
 805            defined within POSIX.1-2008.

806            This Option Group consists of the set of the following options from within POSIX.1-2008 (see  
 807            Section 2.1.6, on page 26):

808            \_POSIX\_ADVISORY\_INFO  
 809            \_POSIX\_CPUTIME  
 810            \_POSIX\_MONOTONIC\_CLOCK  
 811            \_POSIX\_SPAWN  
 812            \_POSIX\_SPORADIC\_SERVER  
 813            \_POSIX\_TYPED\_MEMORY\_OBJECTS

814            If the implementation supports the Advanced Realtime Option Group, then the following  
 815            symbolic constants shall be defined by the implementation to have the value 200809L:

816            \_POSIX\_ADVISORY\_INFO  
 817            \_POSIX\_CPUTIME  
 818            \_POSIX\_MONOTONIC\_CLOCK  
 819            \_POSIX\_SPAWN  
 820            \_POSIX\_SPORADIC\_SERVER  
 821            \_POSIX\_TYPED\_MEMORY\_OBJECTS

822            If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant  
 823            `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the  
 824            value 200809L.

### 825            **Realtime Threads**

826            The Realtime Threads Option Group is denoted by the symbolic constant  
 827            `_XOPEN_REALTIME_THREADS`.

828            This Option Group consists of the set of the following options from within POSIX.1-2008 (see  
 829            Section 2.1.6, on page 26):

830            \_POSIX\_THREAD\_PRIO\_INHERIT  
 831            \_POSIX\_THREAD\_PRIO\_PROTECT  
 832            \_POSIX\_THREAD\_PRIORITY\_SCHEDULING

833            Where applicable, whole pages are marked `REALTIME THREADS`, together with the  
 834            appropriate option margin legend for the SYNOPSIS section (see Section 1.7.1, on page 7).

835            An implementation that claims conformance to this Option Group shall set  
 836            `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

837            If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the  
 838            following options shall also be defined by the implementation to have the value 200809L:

839            \_POSIX\_THREAD\_PRIO\_INHERIT  
 840            \_POSIX\_THREAD\_PRIO\_PROTECT  
 841            \_POSIX\_THREAD\_PRIORITY\_SCHEDULING

### 842            **Advanced Realtime Threads**

843            An implementation that claims conformance to this Option Group shall also support the  
 844            Realtime Threads Option Group.

845            Where entire functions are included in the Option Group, the NAME section is marked with  
 846            `ADVANCED_REALTIME_THREADS`. Where additional semantics have been added to existing  
 847            pages, the new material is identified by use of the appropriate margin legend for the underlying  
 848            option defined within POSIX.1-2008.

849            This Option Group consists of the set of the following options from within POSIX.1-2008 (see  
 850            Section 2.1.6, on page 26):

851            \_POSIX\_THREAD\_CPUTIME  
 852            \_POSIX\_THREAD\_SPORADIC\_SERVER

853            If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value  
 854            200809L, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be  
 855            defined by the implementation to have the value 200809L.

856            If the implementation supports the Advanced Realtime Threads Option Group, then the  
 857            following symbolic constants shall be defined by the implementation to have the value 200809L:

858            \_POSIX\_THREAD\_CPUTIME  
859            \_POSIX\_THREAD\_SPORADIC\_SERVER

## 860            **Tracing**

861            This Option Group includes a set of tracing functions drawn from options within POSIX.1-2008  
862            (see [Section 2.1.6](#), on page 26).

863            Where entire functions are included in the Option Group, the NAME section is marked with  
864            TRACING. Where additional semantics have been added to existing pages, the new material is  
865            identified by use of the appropriate margin legend for the underlying option defined within  
866            POSIX.1-2008.

867            This Option Group consists of the set of the following options from within POSIX.1-2008 (see  
868            [Section 2.1.6](#), on page 26):

869            \_POSIX\_TRACE  
870            \_POSIX\_TRACE\_EVENT\_FILTER  
871            \_POSIX\_TRACE\_LOG  
872            \_POSIX\_TRACE\_INHERIT

873            If the implementation supports the Tracing Option Group, then the following symbolic  
874            constants shall be defined by the implementation to have the value 200809L:

875            \_POSIX\_TRACE  
876            \_POSIX\_TRACE\_EVENT\_FILTER  
877            \_POSIX\_TRACE\_LOG  
878            \_POSIX\_TRACE\_INHERIT

## 879            **XSI STREAMS**

880    OB XSR    This section describes the XSI STREAMS Option Group, denoted by the symbolic constant  
881            \_XOPEN\_STREAMS. The functionality described in this section shall be provided on  
882            implementations that support the XSI STREAMS option (and the rest of this section is not  
883            further shaded).

884            This Option Group includes functionality related to STREAMS, a uniform mechanism for  
885            implementing networking services and other character-based I/O as described in XSH [Section](#)  
886            2.6 (on page 494).

887            It includes the following functions:

888            *fattach()*        *ioctl()*  
889            *fdetach()*       *isastream()*  
890            *getmsg()*        *putmsg()*  
891            *getpmsg()*       *putpmsg()*

892            and the `<stropts.h>` header.

893            Where applicable, whole pages are marked STREAMS, together with the appropriate option  
894            margin legend for the SYNOPSIS section (see [Section 1.7.1](#), on page 7). Where additional  
895            semantics have been added to existing pages, the new material is identified by use of the  
896            appropriate margin legend for the underlying option defined within POSIX.1-2008.

897            An implementation that claims conformance to this Option Group shall set `_XOPEN_STREAMS`  
898            to a value other than `-1`.

899 **2.1.6 Options**

900 The symbolic constants defined in `<unistd.h>`, [Constants for Options and Option Groups](#) (on  
 901 page 430) reflect implementation options for POSIX.1-2008. These symbols can be used by the  
 902 application to determine which of three categories of support for optional facilities are provided  
 903 by the implementation.

## 904 1. Option not supported for compilation.

905 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
 906 with value `-1`, or by leaving it undefined) that the option is not supported for compilation  
 907 and, at the time of compilation, is not supported for runtime use. In this case, the headers,  
 908 data types, function interfaces, and utilities required only for the option need not be  
 909 present. A later runtime check using the `fpathconf()`, `pathconf()`, or `sysconf` functions  
 910 defined in the System Interfaces volume of POSIX.1-2008 or the `getconf` utility defined in  
 911 the Shell and Utilities volume of POSIX.1-2008 can in some circumstances indicate that  
 912 the option is supported at runtime. (For example, an old application binary might be run  
 913 on a newer implementation to which support for the option has been added.)

## 914 2. Option always supported.

915 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
 916 with a value greater than zero) that the option is supported both for compilation and for  
 917 use at runtime. In this case, all headers, data types, function interfaces, and utilities  
 918 required only for the option shall be available and shall operate as specified. Runtime  
 919 checks with `fpathconf()`, `pathconf()`, or `sysconf` shall indicate that the option is supported.

## 920 3. Option might or might not be supported at runtime.

921 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
 922 with value zero) that the option is supported for compilation and might or might not be  
 923 supported at runtime. In this case, the `fpathconf()`, `pathconf()`, or `sysconf()` functions  
 924 defined in the System Interfaces volume of POSIX.1-2008 or the `getconf` utility defined in  
 925 the Shell and Utilities volume of POSIX.1-2008 can be used to retrieve the value of each  
 926 symbol on each specific implementation to determine whether the option is supported at  
 927 runtime. All headers, data types, and function interfaces required to compile and execute  
 928 applications which use the option at runtime (after checking at runtime that the option is  
 929 supported) shall be provided, but if the option is not supported at runtime they need not  
 930 operate as specified. Utilities or other facilities required only for the option, but not  
 931 needed to compile and execute such applications, need not be present.

932 If an option is not supported for compilation, an application that attempts to use anything  
 933 associated only with the option is considered to be requiring an extension. Unless explicitly  
 934 specified otherwise, the behavior of functions associated with an option that is not supported at  
 935 runtime is unspecified, and an application that uses such functions without first checking  
 936 `fpathconf()`, `pathconf()`, or `sysconf` is considered to be requiring an extension.

937 Margin codes are defined for each option (see [Section 1.7.1](#), on page 7).

938 **2.1.6.1 System Interfaces**

939 Refer to `<unistd.h>`, [Constants for Options and Option Groups](#) (on page 430) for the list of  
 940 options.

## Conformance

## Implementation Conformance

## 941 2.1.6.2 Shell and Utilities

942 Each of these symbols shall be considered valid names by the implementation. Refer to  
943 **<unistd.h>**, **Constants for Options and Option Groups** (on page 430).

944 The literal names shown below apply only to the *getconf* utility.

## 945 CD POSIX2\_C\_DEV

946 The system supports the C-Language Development Utilities option.

947 The utilities in the C-Language Development Utilities option are used for the development  
948 of C-language applications, including compilation or translation of C source code and  
949 complex program generators for simple lexical tasks and processing of context-free  
950 grammars.

951 The utilities listed below may be provided by a conforming system; however, any system  
952 claiming conformance to the C-Language Development Utilities option shall provide all of  
953 the utilities listed.

954 *c99*

955 *lex*

956 *yacc*

## 957 POSIX2\_CHAR\_TERM

958 The system supports the Terminal Characteristics option. This value need not be present on  
959 a system not supporting the User Portability Utilities option.

960 Where applicable, the dependency is noted within the description of the utility.

961 This option applies only to systems supporting the User Portability Utilities option. If  
962 supported, then the system supports at least one terminal type capable of all operations  
963 described in POSIX.1-2008; see **Section 10.2** (on page 198).

## 964 FD POSIX2\_FORT\_DEV

965 The system supports the FORTRAN Development Utilities option.

966 The *fort77* FORTRAN compiler is the only utility in the FORTRAN Development Utilities  
967 option. This is used for the development of FORTRAN language applications, including  
968 compilation or translation of FORTRAN source code.

969 The *fort77* utility may be provided by a conforming system; however, any system claiming  
970 conformance to the FORTRAN Development Utilities option shall provide the *fort77* utility.

## 971 FR POSIX2\_FORT\_RUN

972 The system supports the FORTRAN Runtime Utilities option.

973 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.

974 The *asa* utility may be provided by a conforming system; however, any system claiming  
975 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.

## 976 POSIX2\_LOCALEDEF

977 The system supports the Locale Creation Utilities option.

978 If supported, the system supports the creation of locales as described in the *localedef* utility.

979 The *localedef* utility may be provided by a conforming system; however, any system  
980 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*  
981 utility.

982	OB BE	<b>POSIX2_PBS</b>
983		The system supports the Batch Environment Services and Utilities option (see XCU <a href="#">Chapter 3</a> , on page 2375).
984		
985		<b>Note:</b> The Batch Environment Services and Utilities option is a combination of mandatory and optional batch services and utilities. The POSIX_PBS symbolic constant implies the system supports all the mandatory batch services and utilities.
986		
987		
988		<b>POSIX2_PBS_ACCOUNTING</b>
989		The system supports the Batch Accounting option.
990		<b>POSIX2_PBS_CHECKPOINT</b>
991		The system supports the Batch Checkpoint/Restart option.
992		<b>POSIX2_PBS_LOCATE</b>
993		The system supports the Locate Batch Job Request option.
994		<b>POSIX2_PBS_MESSAGE</b>
995		The system supports the Batch Job Message Request option.
996		<b>POSIX2_PBS_TRACK</b>
997		The system supports the Track Batch Job Request option.
998	SD	<b>POSIX2_SW_DEV</b>
999		The system supports the Software Development Utilities option.
1000		The utilities in the Software Development Utilities option are used for the development of applications, including compilation or translation of source code, the creation and maintenance of library archives, and the maintenance of groups of inter-dependent programs.
1001		
1002		
1003		
1004		The utilities listed below may be provided by the conforming system; however, any system claiming conformance to the Software Development Utilities option shall provide all of the utilities listed here.
1005		
1006		
1007		<i>ar</i>
1008		<i>make</i>
1009		<i>nm</i>
1010		<i>strip</i>
1011	UP	<b>POSIX2_UPE</b>
1012		The system supports the User Portability Utilities option.
1013		The utilities in the User Portability Utilities option shall be implemented on all systems that claim conformance to this option, except for the <i>vi</i> utility which is noted as having features that cannot be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the system shall support all such features on at least one terminal type; see <a href="#">Section 10.2</a> (on page 198).
1014		
1015		
1016		
1017		
1018		The list of utilities in the User Portability Utilities option is as follows:
1019		<i>bg fc jobs talk</i>
1020		<i>ex fg more vi</i>
1021	XSI	<b>XOPEN_UNIX</b>
1022		The system supports the X/Open System Interfaces (XSI) option (see <a href="#">Section 2.1.4</a> , on page 19).
1023		

## Conformance

## Implementation Conformance

1024 UU XOPEN\_UUCP  
 1025 The system supports the UUCP Utilities option.  
 1026 The list of utilities in the UUCP Utilities option is as follows:  
 1027 *uucp*  
 1028 *uustat*  
 1029 *uux*

## 1030 2.2 Application Conformance

1031 For the purposes of POSIX.1-2008, the application conformance requirements given in this  
 1032 section apply.

1033 All applications claiming conformance to POSIX.1-2008 shall use only language-dependent  
 1034 services for the C programming language described in Section 2.3 (on page 31), shall use only  
 1035 the utilities and facilities defined in the Shell and Utilities volume of POSIX.1-2008, and shall fall  
 1036 within one of the following categories.

### 1037 2.2.1 Strictly Conforming POSIX Application

1038 A Strictly Conforming POSIX Application is an application that requires only the facilities  
 1039 described in POSIX.1-2008. Such an application:

- 1040 1. Shall accept any implementation behavior that results from actions it takes in areas  
 1041 described in POSIX.1-2008 as *implementation-defined* or *unspecified*, or where POSIX.1-2008  
 1042 indicates that implementations may vary
- 1043 2. Shall not perform any actions that are described as producing *undefined* results
- 1044 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2008,  
 1045 but shall not rely on any value in the range being greater than the minimums listed or  
 1046 being less than the maximums listed in POSIX.1-2008
- 1047 4. Shall not use facilities designated as *obsolescent*
- 1048 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
 1049 facilities whose availability is indicated by Section 2.1.3 (on page 16)
- 1050 6. For the C programming language, shall not produce any output dependent on any  
 1051 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or  
 1052 *implementation-defined*, unless the System Interfaces volume of POSIX.1-2008 specifies the  
 1053 behavior
- 1054 7. For the C programming language, shall not exceed any minimum implementation limit  
 1055 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of  
 1056 POSIX.1-2008 specifies a higher minimum implementation limit
- 1057 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200809L before  
 1058 any header is included

1059 Within POSIX.1-2008, any restrictions placed upon a Conforming POSIX Application shall  
 1060 restrict a Strictly Conforming POSIX Application.

1061 **2.2.2 Conforming POSIX Application**1062 2.2.2.1 *ISO/IEC Conforming POSIX Application*

1063 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities  
 1064 described in POSIX.1-2008 and approved Conforming Language bindings for any ISO or IEC  
 1065 standard. Such an application shall include a statement of conformance that documents all  
 1066 options and limit dependencies, and all other ISO or IEC standards used.

1067 2.2.2.2 *<National Body> Conforming POSIX Application*

1068 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming  
 1069 POSIX Application in that it also may use specific standards of a single ISO/IEC member body  
 1070 referred to here as <National Body>. Such an application shall include a statement of  
 1071 conformance that documents all options and limit dependencies, and all other <National Body>  
 1072 standards used.

1073 **2.2.3 Conforming POSIX Application Using Extensions**

1074 A Conforming POSIX Application Using Extensions is an application that differs from a  
 1075 Conforming POSIX Application only in that it uses non-standard facilities that are consistent  
 1076 with POSIX.1-2008. Such an application shall fully document its requirements for these extended  
 1077 facilities, in addition to the documentation required of a Conforming POSIX Application. A  
 1078 Conforming POSIX Application Using Extensions shall be either an ISO/IEC Conforming  
 1079 POSIX Application Using Extensions or a <National Body> Conforming POSIX Application  
 1080 Using Extensions (see [Section 2.2.2.1](#) and [Section 2.2.2.2](#)).

1081 **2.2.4 Strictly Conforming XSI Application**

1082 A Strictly Conforming XSI Application is an application that requires only the facilities  
 1083 described in POSIX.1-2008. Such an application:

- 1084 1. Shall accept any implementation behavior that results from actions it takes in areas  
 1085 described in POSIX.1-2008 as *implementation-defined* or *unspecified*, or where POSIX.1-2008  
 1086 indicates that implementations may vary
- 1087 2. Shall not perform any actions that are described as producing *undefined* results
- 1088 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2008,  
 1089 but shall not rely on any value in the range being greater than the minimums listed or  
 1090 being less than the maximums listed in POSIX.1-2008
- 1091 4. Shall not use facilities designated as *obsolescent*
- 1092 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
 1093 facilities whose availability is indicated by [Section 2.1.4](#) (on page 19)
- 1094 6. For the C programming language, shall not produce any output dependent on any  
 1095 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*  
 1096 *defined*, unless the System Interfaces volume of POSIX.1-2008 specifies the behavior

*Conformance**Application Conformance*

1097 7. For the C programming language, shall not exceed any minimum implementation limit  
 1098 defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-2008  
 1099 specifies a higher minimum implementation limit

1100 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 700 before any  
 1101 header is included

1102 Within POSIX.1-2008, any restrictions placed upon a Conforming POSIX Application shall  
 1103 restrict a Strictly Conforming XSI Application.

#### 1104 **2.2.5 Conforming XSI Application Using Extensions**

1105 A Conforming XSI Application Using Extensions is an application that differs from a Strictly  
 1106 Conforming XSI Application only in that it uses non-standard facilities that are consistent with  
 1107 POSIX.1-2008. Such an application shall fully document its requirements for these extended  
 1108 facilities, in addition to the documentation required of a Strictly Conforming XSI Application.

### 1109 **2.3 Language-Dependent Services for the C Programming Language**

1110 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX  
 1111 conformance as described in [Section 2.1.3](#) (on page 16).

### 1112 **2.4 Other Language-Related Specifications**

1113 POSIX.1-2008 is currently specified in terms of the shell command language and ISO C. Bindings  
 1114 to other programming languages are being developed.

1115 If conformance to POSIX.1-2008 is claimed for implementation of any programming language,  
 1116 the implementation of that language shall support the use of external symbols distinct to at least  
 1117 31 bytes in length in the source program text. (That is, identifiers that differ at or before the  
 1118 thirty-first byte shall be distinct.) If a national or international standard governing a language  
 1119 defines a maximum length that is less than this value, the language-defined maximum shall be  
 1120 supported. External symbols that differ only by case shall be distinct when the character set in  
 1121 use distinguishes uppercase and lowercase characters and the language permits (or requires)  
 1122 uppercase and lowercase characters to be distinct in external symbols.

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Definitions

1123

1124

1125

1126

1127

For the purposes of POSIX.1-2008, the following terms and definitions apply. The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition should be referenced for terms not defined in this section.

1128

1129

1130

**Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

1131

### 3.1 Abortive Release

1132

An abrupt termination of a network connection that may result in the loss of data.

1133

### 3.2 Absolute Pathname

1134

1135

A pathname beginning with a single or more than two <slash> characters; see also [Section 3.266](#) (on page 75).

1136

**Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).

1137

### 3.3 Access Mode

1138

A particular form of access permitted to a file.

1139

### 3.4 Additional File Access Control Mechanism

1140

1141

1142

An implementation-defined mechanism that is layered upon the access control mechanisms defined here, but which do not grant permissions beyond those defined herein, although they may further restrict them.

1143

**Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

1144

### 3.5 Address Space

1145

The memory locations that can be referenced by a process or the threads of a process.

### 1146 3.6 Advisory Information

1147 An interface that advises the implementation on (portable) application behavior so that it can  
1148 optimize the system.

### 1149 3.7 Affirmative Response

1150 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
1151 keyword **yesexpr**, matching an extended regular expression in the current locale.

1152 **Note:** The *LC\_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 164).

### 1153 3.8 Alert

1154 To cause the user's terminal to give some audible or visual indication that an error or some other  
1155 event has occurred. When the standard output is directed to a terminal device, the method for  
1156 alerting the terminal user is unspecified. When the standard output is not directed to a terminal  
1157 device, the alert is accomplished by writing the alert to standard output (unless the utility  
1158 description indicates that the use of standard output produces undefined results in this case).

### 1159 3.9 Alert Character (<alert>)

1160 A character that in the output stream should cause a terminal to alert its user via a visual or  
1161 audible notification. It is the character designated by '`\a`' in the C language. It is unspecified  
1162 whether this character is the exact sequence transmitted to an output device by the system to  
1163 accomplish the alert function.

### 1164 3.10 Alias Name

1165 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
1166 from the portable character set and any of the following characters: '`!`', '`%`', '`'`', '`,`', '`@`'.

1167 Implementations may allow other characters within alias names as an extension.

1168 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

### 1169 3.11 Alignment

1170 A requirement that objects of a particular type be located on storage boundaries with addresses  
1171 that are particular multiples of a byte address.

1172 **Note:** See also the ISO C standard, Section B3.

### 1173 3.12 Alternate File Access Control Mechanism

1174 An implementation-defined mechanism that is independent of the access control mechanisms  
1175 defined herein, and which if enabled on a file may either restrict or extend the permissions of a  
1176 given user. POSIX.1-2008 defines when such mechanisms can be enabled and when they are  
1177 disabled.

1178 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

### 1179 3.13 Alternate Signal Stack

1180 Memory associated with a thread, established upon request by the implementation for a thread,  
1181 separate from the thread signal stack, in which signal handlers responding to signals sent to that  
1182 thread may be executed.

### 1183 3.14 Ancillary Data

1184 Protocol-specific, local system-specific, or optional information. The information can be both  
1185 local or end-to-end significant, header information, part of a data portion, protocol-specific, and  
1186 implementation or system-specific.

### 1187 3.15 Angle Brackets

1188 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase  
1189 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,  
1190 and '`>`' immediately follows it. When describing these characters in the portable character set,  
1191 the names `<less-than-sign>` and `<greater-than-sign>` are used.

### 1192 3.16 Apostrophe Character (`<apostrophe>`)

1193 The character designated by '`\'`' in the C language, also known as the single-quote character.

### 1194 3.17 Application

1195 A computer program that performs some desired function.

1196 **3.18 Application Address**

1197 Endpoint address of a specific application.

1198 **3.19 Application Program Interface (API)**

1199 The definition of syntax and semantics for providing computer system services.

1200 **3.20 Appropriate Privileges**

1201 An implementation-defined means of associating privileges with a process with regard to the  
 1202 function calls, function call options, and the commands that need special privileges. There may  
 1203 be zero or more such means. These means (or lack thereof) are described in the conformance  
 1204 document.

1205 **Note:** Function calls are defined in the System Interfaces volume of POSIX.1-2008, and commands are  
 1206 defined in the Shell and Utilities volume of POSIX.1-2008.

1207 **3.21 Argument**

1208 In the shell command language, a parameter passed to a utility as the equivalent of a single  
 1209 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,  
 1210 option-arguments, or operands following the command name.

1211 **Note:** The Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213) and XCU [Section](#)  
 1212 [2.9.1.1](#) (on page 2317).

1213 In the C language, an expression in a function call expression or a sequence of preprocessing  
 1214 tokens in a function-like macro invocation.

1215 **3.22 Arm (a Timer)**

1216 To start a timer measuring the passage of time, enabling notifying a process when the specified  
 1217 time or time interval has passed.

1218 **3.23 Asterisk Character (<asterisk>)**

1219 The character ' \* '.

1220 **3.24 Async-Cancel-Safe Function**

1221 A function that may be safely invoked by an application while the asynchronous form of  
 1222 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such.

### 1223 3.25 Asynchronous Events

1224 Events that occur independently of the execution of the application.

### 1225 3.26 Asynchronous Input and Output

1226 A functionality enhancement to allow an application process to queue data input and output  
1227 commands with asynchronous notification of completion.

### 1228 3.27 Async-Signal-Safe Function

1229 A function that may be invoked, without restriction, from signal-catching functions. No function  
1230 is async-signal-safe unless explicitly described as such.

### 1231 3.28 Asynchronously-Generated Signal

1232 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals  
1233 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a  
1234 property of how the signal was generated and not a property of the signal number. All signals  
1235 may be generated asynchronously.

1236 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

### 1237 3.29 Asynchronous I/O Completion

1238 For an asynchronous read or write operation, when a corresponding synchronous read or write  
1239 would have completed and when any associated status fields have been updated.

### 1240 3.30 Asynchronous I/O Operation

1241 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from  
1242 further use of the processor.

1243 This implies that the process and the I/O operation may be running concurrently.

### 1244 3.31 Authentication

1245 The process of validating a user or process to verify that the user or process is not a counterfeit.

1246 **3.32 Authorization**

1247 The process of verifying that a user or process has permission to use a resource in the manner  
1248 requested.

1249 To ensure security, the user or process would also need to be authenticated before granting  
1250 access.

1251 **3.33 Background Job**

1252 See *Background Process Group* in [Section 3.35](#).

1253 **3.34 Background Process**

1254 A process that is a member of a background process group.

1255 **3.35 Background Process Group (or Background Job)**

1256 Any process group, other than a foreground process group, that is a member of a session that  
1257 has established a connection with a controlling terminal.

1258 **3.36 Backquote Character**

1259 The character ' ` ', also known as <grave-accent>.

1260 **3.37 Backslash Character (<backslash>)**

1261 The character designated by ' \ ' in the C language, also known as reverse solidus.

1262 **3.38 Backspace Character (<backspace>)**

1263 A character that, in the output stream, should cause printing (or displaying) to occur one  
1264 column position previous to the position about to be printed. If the position about to be printed  
1265 is at the beginning of the current line, the behavior is unspecified. It is the character designated  
1266 by ' \b ' in the C language. It is unspecified whether this character is the exact sequence  
1267 transmitted to an output device by the system to accomplish the backspace function. The  
1268 backspace defined here is not necessarily the ERASE special character.

1269 **Note:** Special Characters are defined in detail in [Section 11.1.9](#) (on page 203).

1270 **3.39 Barrier**

1271 A synchronization object that allows multiple threads to synchronize at a particular point in  
1272 their execution.

1273 **3.40 Basename**

1274 The final, or only, filename in a pathname.

1275 **3.41 Basic Regular Expression (BRE)**

1276 A regular expression (see [Section 3.315](#), on page 84) used by the majority of utilities that select  
1277 strings from a set of character strings.

1278 **Note:** Basic Regular Expressions are described in detail in [Section 9.3](#) (on page 183).

1279 **3.42 Batch Access List**

1280 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a  
1281 batch queue.

1282 A batch access list is associated with a batch queue. A batch server uses the batch access list of a  
1283 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1284 **3.43 Batch Administrator**

1285 A user that is authorized to modify all the attributes of queues and jobs and to change the status  
1286 of a batch server.

1287 **3.44 Batch Client**

1288 A computational entity that utilizes batch services by making requests of batch servers.

1289 Batch clients often provide the means by which users access batch services, although a batch  
1290 server may act as a batch client by virtue of making requests of another batch server.

**1291 3.45 Batch Destination**

1292 The batch server in a batch system to which a batch job should be sent for processing.

1293 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.  
1294 A batch destination may consist of a batch server-specific portion, a network-wide portion, or  
1295 both. The batch server-specific portion is referred to as the “batch queue”. The network-wide  
1296 portion is referred to as a “batch server name”.

**1297 3.46 Batch Destination Identifier**

1298 A string that identifies a specific batch destination.

1299 A string of characters in the portable character set used to specify a particular batch destination.

1300 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

**1301 3.47 Batch Directive**

1302 A line from a file that is interpreted by the batch server. The line is usually in the form of a  
1303 comment and is an additional means of passing options to the *qsub* utility.

1304 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

**1305 3.48 Batch Job**

1306 A set of computational tasks for a computing system.

1307 Batch jobs are managed by batch servers.

1308 Once created, a batch job may be executing or pending execution. A batch job that is executing  
1309 has an associated session leader (a process) that initiates and monitors the computational tasks  
1310 of the batch job.

**1311 3.49 Batch Job Attribute**

1312 A named data type whose value affects the processing of a batch job.

1313 The values of the attributes of a batch job affect the processing of that job by the batch server that  
1314 manages the batch job.

**1315 3.50 Batch Job Identifier**

1316 A unique name for a batch job. A name that is unique among all other batch job identifiers in a  
1317 batch system and that identifies the batch server to which the batch job was originally  
1318 submitted.

1319 **3.51 Batch Job Name**

1320 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

1321 **3.52 Batch Job Owner**1322 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see  
1323 also [Section 3.429](#), on page 102) and *hostname* is a network host name.1324 **3.53 Batch Job Priority**1325 A value specified by the user that may be used by an implementation to determine the order in  
1326 which batch jobs are selected to be executed. Job priority has a numeric value in the range  
1327 -1 024 to 1 023.1328 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.1329 **3.54 Batch Job State**1330 An attribute of a batch job which determines the types of requests that the batch server that  
1331 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING,  
1332 HELD, WAITING, EXITING, and TRANSFERRING.1333 **3.55 Batch Name Service**1334 A service that assigns batch names that are unique within the batch name space, and that can  
1335 translate a unique batch name into the location of the named batch entity.1336 **3.56 Batch Name Space**

1337 The environment within which a batch name is known to be unique.

1338 **3.57 Batch Node**

1339 A host containing part or all of a batch system.

1340 A batch node is a host meeting at least one of the following conditions:

- 1341 • Capable of executing a batch client
- 1342 • Contains a routing batch queue
- 1343 • Contains an execution batch queue

1344 **3.58 Batch Operator**

1345 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and  
 1346 may change the status of the batch server.

1347 **3.59 Batch Queue**

1348 A manageable object that represents a set of batch jobs and is managed by a single batch server.

1349 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from  
 1350 the batch queue for execution based on attributes such as priority, resource requirements, and  
 1351 hold conditions.

1352 See also XCU [Section 3.1.2](#) (on page 2376).1353 **3.60 Batch Queue Attribute**

1354 A named data type whose value affects the processing of all batch jobs that are members of the  
 1355 batch queue.

1356 A batch queue has attributes that affect the processing of batch jobs that are members of the  
 1357 batch queue.

1358 **3.61 Batch Queue Position**

1359 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.  
 1360 This is defined in part by submission time and priority; see also [Section 3.62](#).

1361 **3.62 Batch Queue Priority**

1362 The maximum job priority allowed for any batch job in a given batch queue.

1363 The batch queue priority is set and may be changed by users with appropriate privileges. The  
 1364 priority is bounded in an implementation-defined manner.

### 1365 3.63 Batch Rerunability

1366 An attribute of a batch job indicating that it may be rerun after an abnormal termination from  
1367 the beginning without affecting the validity of the results.

### 1368 3.64 Batch Restart

1369 The action of resuming the processing of a batch job from the point of the last checkpoint.  
1370 Typically, this is done if the batch job has been interrupted because of a system failure.

### 1371 3.65 Batch Server

1372 A computational entity that provides batch services.

### 1373 3.66 Batch Server Name

1374 A string of characters in the portable character set used to specify a particular server in a  
1375 network.

1376 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

### 1377 3.67 Batch Service

1378 Computational and organizational services performed by a batch system on behalf of batch jobs.

1379 Batch services are of two types: requested and deferred.

1380 **Note:** Batch Services are listed in XCU [Table 3-5](#) (on page 2390).

### 1381 3.68 Batch Service Request

1382 A solicitation of services from a batch client to a batch server.

1383 A batch service request may entail the exchange of any number of messages between the batch  
1384 client and the batch server.

1385 When naming specific types of service requests, the term "request" is qualified by the type of  
1386 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

### 1387 3.69 Batch Submission

1388 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*  
1389 *Request* to perform a specified computational task.

1390 **3.70 Batch System**

1391 A collection of one or more batch servers.

1392 **3.71 Batch Target User**

1393 The name of a user on the batch destination batch server.

1394 The target user is the user name under whose account the batch job is to execute on the  
1395 destination batch server.

1396 **3.72 Batch User**

1397 A user who is authorized to make use of batch services.

1398 **3.73 Bind**

1399 The process of assigning a network address to an endpoint.

1400 **3.74 Blank Character (<blank>)**

1401 One of the characters that belong to the **blank** character class as defined via the *LC\_CTYPE*  
1402 category in the current locale. In the POSIX locale, a <blank> character is either a <tab> or a  
1403 <space>.

1404 **3.75 Blank Line**

1405 A line consisting solely of zero or more <blank> characters terminated by a <newline>; see also  
1406 [Section 3.145](#) (on page 56).

1407 **3.76 Blocked Process (or Thread)**

1408 A process (or thread) that is waiting for some condition (other than the availability of a  
1409 processor) to be satisfied before it can continue execution.

1410 **3.77 Blocking**

1411 A property of an open file description that causes function calls associated with it to wait for the  
1412 requested action to be performed before returning.

1413 **3.78 Block-Mode Terminal**

1414 A terminal device operating in a mode incapable of the character-at-a-time input and output  
 1415 operations described by some of the standard utilities.

1416 **Note:** Output Devices and Terminal Types are defined in detail in [Section 10.2](#) (on page 198).

1417 **3.79 Block Special File**

1418 A file that refers to a device. A block special file is normally distinguished from a character  
 1419 special file by providing access to the device in a manner such that the hardware characteristics  
 1420 of the device are not visible.

1421 **3.80 Braces**

1422 The characters ' {' (left-curly-bracket) and ' }' (right-curly-bracket). When used in the phrase  
 1423 "enclosed in (curly) braces" the symbol ' {' immediately precedes the object to be enclosed, and  
 1424 ' }' immediately follows it. When describing these characters in the portable character set, the  
 1425 names <left-curly-bracket> and <right-curly-bracket> are used for ' {' and ' }', and <left-brace> and  
 1426 <right-brace> are used for ' {' and ' }'.

1427 **3.81 Brackets**

1428 The characters ' [' (left-square-bracket) and ' ]' (right-square-bracket). When used in the  
 1429 phrase "enclosed in (square) brackets" the symbol ' [' immediately precedes the object to be  
 1430 enclosed, and ' ]' immediately follows it. When describing these characters in the portable  
 1431 character set, the names <left-square-bracket> and <right-square-bracket> are used.

1432 **3.82 Broadcast**

1433 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and  
 1434 RFC 922.

1435 **3.83 Built-In Utility (or Built-In)**

1436 A utility implemented within a shell. The utilities referred to as special built-ins have special  
 1437 qualities. Unless qualified, the term “built-in” includes the special built-in utilities. Regular  
 1438 built-ins are not required to be actually built into the shell on the implementation, but they do  
 1439 have special command-search qualities.

1440 **Note:** Special Built-In Utilities are defined in detail in XCU [Section 2.14](#) (on page 2334).

1441 Regular Built-In Utilities are defined in detail in XCU [Section 2.9.1.1](#) (on page 2317).

1442 **3.84 Byte**

1443 An individually addressable unit of data storage that is exactly an octet, used to store a character  
 1444 or a portion of a character; see also [Section 3.87](#) (on page 47). A byte is composed of a  
 1445 contiguous sequence of 8 bits. The least significant bit is called the “low-order” bit; the most  
 1446 significant is called the “high-order” bit.

1447 **Note:** The definition of byte from the ISO C standard is broader than the above and might  
 1448 accommodate hardware architectures with different sized addressable units than octets.

1449 **3.85 Byte Input/Output Functions**

1450 The functions that perform byte-oriented input from streams or byte-oriented output to streams:  
 1451 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *getdelim()*,  
 1452 *getline()*, *gets()*, *printf()*, *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1453 **Note:** Functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

1454 **3.86 Carriage-Return Character (<carriage-return>)**

1455 A character that in the output stream indicates that printing should start at the beginning of the  
 1456 same physical line in which the carriage-return occurred. It is the character designated by ‘\r’  
 1457 in the C language. It is unspecified whether this character is the exact sequence transmitted to an  
 1458 output device by the system to accomplish the movement to the beginning of the line.

1459 **3.87 Character**

1460 A sequence of one or more bytes representing a single graphic symbol or control code.

1461 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte  
 1462 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,  
 1463 *character* here has no necessary relationship with storage space, and *byte* is used when storage  
 1464 space is discussed.

1465 See the definition of the portable character set in [Section 6.1](#) (on page 125) for a further  
 1466 explanation of the graphical representations of (abstract) characters, as opposed to character  
 1467 encodings.

1468 **3.88 Character Array**1469 An array of elements of type **char**.1470 **3.89 Character Class**

1471 A named set of characters sharing an attribute associated with the name of the class. The classes  
 1472 and the characters that they contain are dependent on the value of the *LC\_CTYPE* category in  
 1473 the current locale.

1474 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

1475 **3.90 Character Set**

1476 A finite set of different characters used for the representation, organization, or control of data.

1477 **3.91 Character Special File**

1478 A file that refers to a device (such as a terminal device file) or that has special properties (such as  
 1479 **/dev/null**).

1480 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

1481 **3.92 Character String**

1482 A contiguous sequence of characters terminated by and including the first null byte.

1483 **3.93 Child Process**

1484 A new process created (by *fork()*, *posix\_spawn()*, or *posix\_spawnp()*) by a given process. A child  
1485 process remains the child of the creating process as long as both processes continue to exist.

1486 **Note:** The *fork()*, *posix\_spawn()*, and *posix\_spawnp()* functions are defined in detail in the System  
1487 Interfaces volume of POSIX.1-2008.

1488 **3.94 Circumflex Character (<circumflex>)**

1489 The character ' ^ '.

1490 **3.95 Clock**

1491 A software or hardware object that can be used to measure the apparent or actual passage of  
1492 time.

1493 The current value of the time measured by a clock can be queried and, possibly, set to a value  
1494 within the legal range of the clock.

1495 **3.96 Clock Jump**

1496 The difference between two successive distinct values of a clock, as observed from the  
1497 application via one of the "get time" operations.

1498 **3.97 Clock Tick**

1499 An interval of time; an implementation-defined number of these occur each second. Clock ticks  
1500 are one of the units that may be used to express a value found in type **clock\_t**.

1501 **3.98 Coded Character Set**

1502 A set of unambiguous rules that establishes a character set and the one-to-one relationship  
1503 between each character of the set and its bit representation.

### 1504 3.99 Codeset

1505 The result of applying rules that map a numeric code value to each element of a character set.  
 1506 An element of a character set may be related to more than one numeric code value but the  
 1507 reverse is not true. However, for state-dependent encodings the relationship between numeric  
 1508 code values and elements of a character set may be further controlled by state information. The  
 1509 character set may contain fewer elements than the total number of possible numeric code values;  
 1510 that is, some code values may be unassigned.

1511 **Note:** Character Encoding is defined in detail in [Section 6.2](#) (on page 128).

### 1512 3.100 Collating Element

1513 The smallest entity used to determine the logical ordering of character or wide-character strings;  
 1514 see also [Section 3.102](#). A collating element consists of either a single character, or two or more  
 1515 characters collating as a single entity. The value of the *LC\_COLLATE* category in the current  
 1516 locale determines the current set of collating elements.

### 1517 3.101 Collation

1518 The logical ordering of character or wide-character strings according to defined precedence  
 1519 rules. These rules identify a collation sequence between the collating elements, and such  
 1520 additional rules that can be used to order strings consisting of multiple collating elements.

### 1521 3.102 Collation Sequence

1522 The relative order of collating elements as determined by the setting of the *LC\_COLLATE*  
 1523 category in the current locale. The collation sequence is used for sorting and is determined from  
 1524 the collating weights assigned to each collating element. In the absence of weights, the collation  
 1525 sequence is the order in which collating elements are specified between **order\_start** and  
 1526 **order\_end** keywords in the *LC\_COLLATE* category.

1527 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to  
 1528 the limit {*COLL\_WEIGHTS\_MAX*}. On each level, elements may be given the same weight (at  
 1529 the primary level, called an equivalence class; see also [Section 3.151](#), on page 57) or be omitted  
 1530 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)  
 1531 are then compared using the next assigned weight (secondary ordering), and so on.

1532 **Note:** {*COLL\_WEIGHTS\_MAX*} is defined in detail in [<limits.h>](#).

1533 **3.103 Column Position**

1534 A unit of horizontal measure related to characters in a line.

1535 It is assumed that each character in a character set has an intrinsic column width independent of  
 1536 any output device. Each printable character in the portable character set has a column width of  
 1537 one. The standard utilities, when used as described in POSIX.1-2008, assume that all characters  
 1538 have integral column widths. The column width of a character is not necessarily related to the  
 1539 internal representation of the character (numbers of bits or bytes).

1540 The column position of a character in a line is defined as one plus the sum of the column widths  
 1541 of the preceding characters in the line. Column positions are numbered starting from 1.

1542 **3.104 Command**

1543 A directive to the shell to perform a particular task.

1544 **Note:** Shell Commands are defined in detail in XCU Section 2.9 (on page 2316).1545 **3.105 Command Language Interpreter**

1546 An interface that interprets sequences of text input as commands. It may operate on an input  
 1547 stream or it may interactively prompt and read commands from a terminal. It is possible for  
 1548 applications to invoke utilities through a number of interfaces, which are collectively considered  
 1549 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*  
 1550 function, although *popen()* and the various forms of *exec* may also be considered to behave as  
 1551 interpreters.

1552 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

1553 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume  
 1554 of POSIX.1-2008.

1555 **3.106 Composite Graphic Symbol**

1556 A graphic symbol consisting of a combination of two or more other graphic symbols in a single  
 1557 character position, such as a diacritical mark and a base character.

1558 **3.107 Condition Variable**

1559 A synchronization object which allows a thread to suspend execution, repeatedly, until some  
 1560 associated predicate becomes true. A thread whose execution is suspended on a condition  
 1561 variable is said to be blocked on the condition variable.

1562 **3.108 Connected Socket**

1563 A connection-mode socket for which a connection has been established, or a connectionless-  
 1564 mode socket for which a peer address has been set. See also [Section 3.109](#), [Section 3.110](#), [Section](#)  
 1565 [3.111](#), and [Section 3.348](#) (on page 89).

1566 **3.109 Connection**

1567 An association established between two or more endpoints for the transfer of data

1568 **3.110 Connection Mode**

1569 The transfer of data in the context of a connection; see also [Section 3.111](#).

1570 **3.111 Connectionless Mode**

1571 The transfer of data other than in the context of a connection; see also [Section 3.110](#) and [Section](#)  
 1572 [3.124](#) (on page 53).

1573 **3.112 Control Character**

1574 A character, other than a graphic character, that affects the recording, processing, transmission,  
 1575 or interpretation of text.

1576 **3.113 Control Operator**

1577 In the shell command language, a token that performs a control function. It is one of the  
 1578 following symbols:

1579 & && ( ) ; ;; newline | ||

1580 The end-of-input indicator used internally by the shell is also considered a control operator.

1581 **Note:** Token Recognition is defined in detail in XCU [Section 2.3](#) (on page 2299).

1582 **3.114 Controlling Process**

1583 The session leader that established the connection to the controlling terminal. If the terminal  
 1584 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be  
 1585 the controlling process.

**1586 3.115 Controlling Terminal**

1587 A terminal that is associated with a session. Each session may have at most one controlling  
1588 terminal associated with it, and a controlling terminal is associated with exactly one session.  
1589 Certain input sequences from the controlling terminal cause signals to be sent to all processes in  
1590 the foreground process group associated with the controlling terminal.

1591 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

**1592 3.116 Conversion Descriptor**

1593 A per-process unique value used to identify an open codeset conversion.

**1594 3.117 Core File**

1595 A file of unspecified format that may be generated when a process terminates abnormally.

**1596 3.118 CPU Time (Execution Time)**

1597 The time spent executing a process or thread, including the time spent executing system services  
1598 on behalf of that process or thread. If the Threads option is supported, then the value of the  
1599 CPU-time clock for a process is implementation-defined. With this definition the sum of all the  
1600 execution times of all the threads in a process might not equal the process execution time, even  
1601 in a single-threaded process, because implementations may differ in how they account for time  
1602 during context switches or for other reasons.

**1603 3.119 CPU-Time Clock**

1604 A clock that measures the execution time of a particular process or thread.

**1605 3.120 CPU-Time Timer**

1606 A timer attached to a CPU-time clock.

**1607 3.121 Current Job**

1608 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There  
1609 is at most one current job; see also [Section 3.203](#) (on page 65).

1610 **3.122 Current Working Directory**

1611 See *Working Directory* in [Section 3.439](#) (on page 104).

1612 **3.123 Cursor Position**

1613 The line and column position on the screen denoted by the terminal's cursor.

1614 **3.124 Datagram**

1615 A unit of data transferred from one endpoint to another in connectionless mode service.

1616 **3.125 Data Segment**

1617 Memory associated with a process, that can contain dynamically allocated data.

1618 **3.126 Deferred Batch Service**

1619 A service that is performed as a result of events that are asynchronous with respect to requests.

1620 **Note:** Once a batch job has been created it is subject to deferred services.

1621 **3.127 Device**

1622 A computer peripheral or an object that appears to the application as such.

1623 **3.128 Device ID**

1624 A non-negative integer used to identify a device.

1625 **3.129 Directory**

1626 A file that contains directory entries. No two directory entries in the same directory have the  
1627 same name.

1628 **3.130 Directory Entry (or Link)**

1629 An object that associates a filename with a file. Several directory entries can associate names

1630 with the same file.

### 1631 3.131 Directory Stream

1632 A sequence of all the directory entries in a particular directory. An open directory stream may be  
1633 implemented using a file descriptor.

### 1634 3.132 Disarm (a Timer)

1635 To stop a timer from measuring the passage of time, disabling any future process notifications  
1636 (until the timer is armed again).

### 1637 3.133 Display

1638 To output to the user's terminal. If the output is not directed to a terminal, the results are  
1639 undefined.

### 1640 3.134 Display Line

1641 A line of text on a physical device or an emulation thereof. Such a line will have a maximum  
1642 number of characters which can be presented.

1643 **Note:** This may also be written as "line on the display".

### 1644 3.135 Dollar-Sign Character (<dollar-sign>)

1645 The character '\$'.

### 1646 3.136 Dot

1647 In the context of naming files, the filename consisting of a single dot character ('.').

1648 **Note:** In the context of shell special built-in utilities, see *dot* in XCU Section 2.14 (on page 2334).

1649 Pathname Resolution is defined in detail in Section 4.12 (on page 111).

1650 **3.137 Dot-Dot**

1651 The filename consisting solely of two dot characters (" . . ").

1652 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).

1653 **3.138 Double-Quote Character**

1654 The character ' " ', also known as <quotation-mark>.

1655 **Note:** The "double" adjective in this term refers to the two strokes in the character glyph.  
1656 POSIX.1-2008 never uses the term "double-quote" to refer to two apostrophes or quotation-  
1657 marks.

1658 **3.139 Downshifting**

1659 The conversion of an uppercase character that has a single character lowercase representation  
1660 into this lowercase representation.

1661 **3.140 Driver**

1662 A module that controls data transferred to and received from devices.

1663 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are  
1664 frequently written separately from the writing of the implementation. A driver may contain  
1665 processor-specific code, and therefore be non-portable.

1666 **3.141 Effective Group ID**

1667 An attribute of a process that is used in determining various permissions, including file access  
1668 permissions; see also [Section 3.188](#) (on page 63).

1669 **3.142 Effective User ID**

1670 An attribute of a process that is used in determining various permissions, including file access  
1671 permissions; see also [Section 3.428](#) (on page 102).

1672 **3.143 Eight-Bit Transparency**

1673 The ability of a software component to process 8-bit characters without modifying or utilizing  
1674 any part of the character in a way that is inconsistent with the rules of the current coded  
1675 character set.

**1676 3.144 Empty Directory**

1677 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link  
1678 to it (other than its own dot entry, if one exists), in dot-dot. No other links to the directory may  
1679 exist. It is unspecified whether an implementation can ever consider the root directory to be  
1680 empty.

**1681 3.145 Empty Line**

1682 A line consisting of only a <newline>; see also [Section 3.75](#) (on page 44).

**1683 3.146 Empty String (or Null String)**

1684 A string whose first byte is a null byte.

**1685 3.147 Empty Wide-Character String**

1686 A wide-character string whose first element is a null wide-character code.

**1687 3.148 Encoding Rule**

1688 The rules used to convert between wide-character codes and multi-byte character codes.

1689 **Note:** Stream Orientation and Encoding Rules are defined in detail in XSH [Section 2.5.2](#) (on page 493).

**1690 3.149 Entire Regular Expression**

1691 The concatenated set of one or more basic regular expressions or extended regular expressions  
1692 that make up the pattern specified for string selection.

1693 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

1694 **3.150 Epoch**

1695 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time  
 1696 (UTC).

1697 **Note:** See also *Seconds Since the Epoch* defined in [Section 4.15](#) (on page 113).

1698 **3.151 Equivalence Class**

1699 A set of collating elements with the same primary collation weight.

1700 Elements in an equivalence class are typically elements that naturally group together, such as all  
 1701 accented letters based on the same base letter.

1702 The collation order of elements within an equivalence class is determined by the weights  
 1703 assigned on any subsequent levels after the primary weight.

1704 **3.152 Era**

1705 A locale-specific method for counting and displaying years.

1706 **Note:** The *LC\_TIME* category is defined in detail in [Section 7.3.5](#) (on page 158).

1707 **3.153 Event Management**

1708 The mechanism that enables applications to register for and be made aware of external events  
 1709 such as data becoming available for reading.

1710 **3.154 Executable File**

1711 A regular file acceptable as a new process image file by the equivalent of the *exec* family of  
 1712 functions, and thus usable as one form of a utility. The standard utilities described as compilers  
 1713 can produce executable files, but other unspecified methods of producing executable files may  
 1714 also be provided. The internal format of an executable file is unspecified, but a conforming  
 1715 application cannot assume an executable file is a text file.

1716 **3.155 Execute**

1717 To perform command search and execution actions, as defined in the Shell and Utilities volume  
 1718 of POSIX.1-2008; see also [Section 3.200](#) (on page 65).

1719 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2317).

1720 **3.156 Execution Time**

1721 See *CPU Time* in [Section 3.118](#) (on page 52).

1722 **3.157 Execution Time Monitoring**

1723 A set of execution time monitoring primitives that allow online measuring of thread and process  
 1724 execution times.

1725 **3.158 Expand**

1726 In the shell command language, when not qualified, the act of applying word expansions.

1727 **Note:** Word Expansions are defined in detail in XCU [Section 2.6](#) (on page 2305).

1728 **3.159 Extended Regular Expression (ERE)**

1729 A regular expression (see also [Section 3.315](#), on page 84) that is an alternative to the Basic  
 1730 Regular Expression using a more extensive syntax, occasionally used by some utilities.

1731 **Note:** Extended Regular Expressions are described in detail in [Section 9.4](#) (on page 188).

1732 **3.160 Extended Security Controls**

1733 Implementation-defined security controls allowed by the file access permission and appropriate  
 1734 privileges (see also [Section 3.20](#), on page 36) mechanisms, through which an implementation can  
 1735 support different security policies from those described in POSIX.1-2008.

1736 **Note:** See also *Extended Security Controls* defined in [Section 4.3](#) (on page 107).

1737 File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

1738 **3.161 Feature Test Macro**

1739 A macro used to determine whether a particular set of features is included from a header.

1740 **Note:** See also XSH [Section 2.2](#) (on page 468).1741 **3.162 Field**1742 In the shell command language, a unit of text that is the result of parameter expansion,  
1743 arithmetic expansion, command substitution, or field splitting. During command processing, the  
1744 resulting fields are used as the command name and its arguments.1745 **Note:** Parameter Expansion is defined in detail in XCU [Section 2.6.2](#) (on page 2306).1746 Arithmetic Expansion is defined in detail in XCU [Section 2.6.4](#) (on page 2310).1747 Command Substitution is defined in detail in XCU [Section 2.6.3](#) (on page 2309).1748 Field Splitting is defined in detail in XCU [Section 2.6.5](#) (on page 2311).1749 For further information on command processing, see XCU [Section 2.9.1](#) (on page 2316).1750 **3.163 FIFO Special File (or FIFO)**

1751 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1752 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of POSIX.1-2008,  
1753 *lseek()*, *open()*, *read()*, and *write()*.1754 **3.164 File**1755 An object that can be written to, or read from, or both. A file has certain attributes, including  
1756 access permissions and type. File types include regular file, character special file, block special  
1757 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported  
1758 by the implementation.1759 **3.165 File Description**1760 See *Open File Description* in [Section 3.253](#) (on page 73).

1761 **3.166 File Descriptor**

1762 A per-process unique, non-negative integer used to identify an open file for the purpose of file  
 1763 access. The value of a file descriptor is from zero to {OPEN\_MAX}. A process can have no more  
 1764 than {OPEN\_MAX} file descriptors open simultaneously. File descriptors may also be used to  
 1765 implement message catalog descriptors and directory streams; see also [Section 3.253](#) (on page  
 1766 73).

1767 **Note:** {OPEN\_MAX} is defined in detail in [<limits.h>](#).

1768 **3.167 File Group Class**

1769 The property of a file indicating access permissions for a process related to the group  
 1770 identification of a process. A process is in the file group class of a file if the process is not in the  
 1771 file owner class and if the effective group ID or one of the supplementary group IDs of the  
 1772 process matches the group ID associated with the file. Other members of the class may be  
 1773 implementation-defined.

1774 **3.168 File Mode**

1775 An object containing the file mode bits and file type of a file.

1776 **Note:** File mode bits and file types are defined in detail in [<sys/stat.h>](#).

1777 **3.169 File Mode Bits**

1778 A file's file permission bits, set-user-ID-on-execution bit (S\_ISUID), set-group-ID-on-execution  
 1779 bit (S\_ISGID), and, on directories, the restricted deletion flag bit (S\_ISVTX).

1780 **Note:** File Mode Bits are defined in detail in [<sys/stat.h>](#).

1781 **3.170 Filename**

1782 A name consisting of 1 to {NAME\_MAX} bytes used to name a file. The characters composing  
 1783 the name may be selected from the set of all character values excluding the <slash> character  
 1784 and the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes  
 1785 referred to as a "pathname component".

1786 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).

1787 **3.171 File Offset**

1788 The byte position in the file where the next I/O operation begins. Each open file description  
 1789 associated with a regular file, block special file, or directory has a file offset. A character special

1790 file that does not refer to a terminal device may have a file offset. There is no file offset specified  
1791 for a pipe or FIFO.

### 1792 **3.172 File Other Class**

1793 The property of a file indicating access permissions for a process related to the user and group  
1794 identification of a process. A process is in the file other class of a file if the process is not in the  
1795 file owner class or file group class.

### 1796 **3.173 File Owner Class**

1797 The property of a file indicating access permissions for a process related to the user  
1798 identification of a process. A process is in the file owner class of a file if the effective user ID of  
1799 the process matches the user ID of the file.

### 1800 **3.174 File Permission Bits**

1801 Information about a file that is used, along with other information, to determine whether a  
1802 process has read, write, or execute/search permission to a file. The bits are divided into three  
1803 parts: owner, group, and other. Each part is used with the corresponding file class of processes.  
1804 These bits are contained in the file mode.

1805 **Note:** File modes are defined in detail in [<sys/stat.h>](#).

1806 File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

### 1807 **3.175 File Serial Number**

1808 A per-file system unique identifier for a file.

### 1809 **3.176 File System**

1810 A collection of files and certain of their attributes. It provides a name space for file serial  
1811 numbers referring to those files.

### 1812 **3.177 File Type**

1813 See *File* in [Section 3.164](#) (on page 59).

1814 **3.178 Filter**

1815 A command whose operation consists of reading data from standard input or a list of input files  
 1816 and writing data to standard output. Typically, its function is to perform some transformation  
 1817 on the data stream.

1818 **3.179 First Open (of a File)**

1819 When a process opens a file that is not currently an open file within any process.

1820 **3.180 Flow Control**

1821 The mechanism employed by a communications provider that constrains a sending entity to  
 1822 wait until the receiving entities can safely receive additional data without loss.

1823 **3.181 Foreground Job**

1824 See *Foreground Process Group* in [Section 3.183](#).

1825 **3.182 Foreground Process**

1826 A process that is a member of a foreground process group.

1827 **3.183 Foreground Process Group (or Foreground Job)**

1828 A process group whose member processes have certain privileges, denied to processes in  
 1829 background process groups, when accessing their controlling terminal. Each session that has  
 1830 established a connection with a controlling terminal has at most one process group of the session  
 1831 as the foreground process group of that controlling terminal.

1832 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#).

1833 **3.184 Foreground Process Group ID**

1834 The process group ID of the foreground process group.

1835 **3.185 Form-Feed Character (<form-feed>)**

1836 A character that in the output stream indicates that printing should start on the next page of an  
 1837 output device. It is the character designated by '`\f`' in the C language. If the form-feed is not  
 1838 the first character of an output line, the result is unspecified. It is unspecified whether this  
 1839 character is the exact sequence transmitted to an output device by the system to accomplish the  
 1840 movement to the next page.

1841 **3.186 Graphic Character**

1842 A member of the **graph** character class of the current locale.

1843 **Note:** The **graph** character class is defined in detail in [Section 7.3.1](#) (on page 139).

1844 **3.187 Group Database**

1845 A system database that contains at least the following information for each group ID:

- 1846 • Group name
- 1847 • Numerical group ID
- 1848 • List of users allowed in the group

1849 The list of users allowed in the group is used by the *newgrp* utility.

1850 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

1851 **3.188 Group ID**

1852 A non-negative integer, which can be contained in an object of type **gid\_t**, that is used to identify  
 1853 a group of system users. Each system user is a member of at least one group. When the identity  
 1854 of a group is associated with a process, a group ID value is referred to as a real group ID, an  
 1855 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

1856 **3.189 Group Name**

1857 A string that is used to identify a group; see also [Section 3.187](#). To be portable across conforming  
 1858 systems, the value is composed of characters from the portable filename character set. The  
 1859 `<hyphen>` should not be used as the first character of a portable group name.

1860 **3.190 Hard Limit**

1861 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.  
 1862 A non-privileged process is restricted to only lowering its hard limit.

**1863 3.191 Hard Link**

1864 The relationship between two directory entries that represent the same file; see also [Section 3.130](#)  
1865 (on page 53). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*  
1866 function. This term is contrasted against symbolic link; see also [Section 3.373](#) (on page 94).

**1867 3.192 Home Directory**

1868 The directory specified by the *HOME* environment variable.

**1869 3.193 Host Byte Order**

1870 The arrangement of bytes in any integer type when using a specific machine architecture.

1871 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a format  
1872 for storage of binary data in which the most significant byte is placed first, with the rest in  
1873 descending order. Little-endian is a format for storage or transmission of binary data in which  
1874 the least significant byte is placed first, with the rest in ascending order. See also [Section 4.9](#) (on  
1875 page 110).

**1876 3.194 Incomplete Line**

1877 A sequence of one or more non-`<newline>` characters at the end of the file.

**1878 3.195 Inf**

1879 A value representing `+infinity` or a value representing `-infinity` that can be stored in a floating  
1880 type. Not all systems support the *Inf* values.

**1881 3.196 Instrumented Application**

1882 An application that contains at least one call to the trace point function *posix\_trace\_event()*. Each  
1883 process of an instrumented application has a mapping of trace event names to trace event type  
1884 identifiers. This mapping is used by the trace stream that is created for that process.

**1885 3.197 Interactive Shell**

1886 A processing mode of the shell that is suitable for direct user interaction.

1887 **3.198 Internationalization**

1888 The provision within a computer program of the capability of making itself adaptable to the  
1889 requirements of different native languages, local customs, and coded character sets.

1890 **3.199 Interprocess Communication**

1891 A functionality enhancement to add a high-performance, deterministic interprocess  
1892 communication facility for local communication.

1893 **3.200 Invoke**

1894 To perform command search and execution actions, except that searching for shell functions and  
1895 special built-in utilities is suppressed; see also [Section 3.155](#) (on page 58).

1896 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2317).

1897 **3.201 Job**

1898 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all  
1899 in the same process group.

1900 **Note:** See also XCU [Section 2.9.2](#) (on page 2318).

1901 **3.202 Job Control**

1902 A facility that allows users selectively to stop (suspend) the execution of processes and continue  
1903 (resume) their execution at a later point. The user typically employs this facility via the  
1904 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

1905 **3.203 Job Control Job ID**

1906 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in  
1907 the following table:

1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916

**Table 3-1** Job Control Job ID Formats

Job Control Job ID	Meaning
%%	Current job.
%+	Current job.
%-	Previous job.
%n	Job number <i>n</i> .
%string	Job whose command begins with <i>string</i> .
%?string	Job whose command contains <i>string</i> .

1917 **3.204 Last Close (of a File)**

1918 When a process closes a file, resulting in the file not being an open file within any process.

1919 **3.205 Line**

1920 A sequence of zero or more non-`<newline>` characters plus a terminating `<newline>` character.

1921 **3.206 Linger**

1922 The period of time before terminating a connection, to allow outstanding data to be transferred.

1923 **3.207 Link**

1924 See *Directory Entry* in [Section 3.130](#) (on page 53).

1925 **3.208 Link Count**

1926 The number of directory entries that refer to a particular file.

1927 **3.209 Local Customs**

1928 The conventions of a geographical area or territory for such things as date, time, and currency  
1929 formats.

1930 **3.210 Local Interprocess Communication (Local IPC)**

1931 The transfer of data between processes in the same system.

1932 **3.211 Locale**

1933 The definition of the subset of a user's environment that depends on language and cultural  
 1934 conventions.

1935 **Note:** Locales are defined in detail in [Chapter 7](#) (on page 135).

1936 **3.212 Localization**

1937 The process of establishing information within a computer system specific to the operation of  
 1938 particular native languages, local customs, and coded character sets.

1939 **3.213 Login**

1940 The unspecified activity by which a user gains access to the system. Each login is associated  
 1941 with exactly one login name.

1942 **3.214 Login Name**

1943 A user name that is associated with a login.

1944 **3.215 Map**

1945 To create an association between a page-aligned range of the address space of a process and  
 1946 some memory object, such that a reference to an address in that range of the address space  
 1947 results in a reference to the associated memory object. The mapped memory object is not  
 1948 necessarily memory-resident.

1949 **3.216 Marked Message**

1950 A STREAMS message on which a certain flag is set. Marking a message gives the application  
 1951 protocol-specific information. An application can use *ioctl()* to determine whether a given  
 1952 message is marked.

1953 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

1954 **3.217 Matched**

1955 A state applying to a sequence of zero or more characters when the characters in the sequence  
 1956 correspond to a sequence of characters defined by a basic regular expression or extended regular  
 1957 expression pattern.

1958 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

1959 **3.218 Memory Mapped Files**

1960 A facility to allow applications to access files as part of the address space.

1961 **3.219 Memory Object**

1962 One of:

- 1963 • A file (see [Section 3.164](#), on page 59)
- 1964 • A shared memory object (see [Section 3.340](#), on page 88)
- 1965 • A typed memory object (see [Section 3.421](#), on page 101)

1966 When used in conjunction with *mmap()*, a memory object appears in the address space of the  
 1967 calling process.

1968 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

1969 **3.220 Memory-Resident**

1970 The process of managing the implementation in such a way as to provide an upper bound on  
 1971 memory access times.

1972 **3.221 Message**

1973 In the context of programmatic message passing, information that can be transferred between  
 1974 processes or threads by being added to and removed from a message queue. A message consists  
 1975 of a fixed-size message buffer.

1976 **3.222 Message Catalog**

1977 In the context of providing natural language messages to the user, a file or storage area  
 1978 containing program messages, command prompts, and responses to prompts for a particular  
 1979 native language, territory, and codeset.

1980 **3.223 Message Catalog Descriptor**

1981 In the context of providing natural language messages to the user, a per-process unique value  
 1982 used to identify an open message catalog. A message catalog descriptor may be implemented  
 1983 using a file descriptor.

1984 **3.224 Message Queue**

1985 In the context of programmatic message passing, an object to which messages can be added and  
 1986 removed. Messages may be removed in the order in which they were added or in priority order.

1987 **3.225 Mode**

1988 A collection of attributes that specifies a file's type and its access permissions.

1989 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

1990 **3.226 Monotonic Clock**

1991 A clock measuring real time, whose value cannot be set via `clock_settime()` and which cannot  
 1992 have negative clock jumps.

1993 **3.227 Mount Point**

1994 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs  
 1995 from that of its parent directory.

1996 **Note:** The `stat` structure is defined in detail in [<sys/stat.h>](#).

1997 **3.228 Multi-Character Collating Element**

1998 A sequence of two or more characters that collate as an entity. For example, in some coded  
 1999 character sets, an accented character is represented by a non-spacing accent, followed by the  
 2000 letter. Other examples are the Spanish elements *ch* and *ll*.

2001 **3.229 Mutex**

2002 A synchronization object used to allow multiple threads to serialize their access to shared data.  
 2003 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has  
 2004 locked a mutex becomes its owner and remains the owner until that same thread unlocks the  
 2005 mutex.

2006 **3.230 Name**

2007 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
 2008 from the portable character set. The first character of a name is not a digit.

2009 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

2010 **3.231 Named STREAM**

2011 A STREAMS-based file descriptor that is attached to a name in the file system name space. All  
 2012 subsequent operations on the named STREAM act on the STREAM that was associated with the  
 2013 file descriptor until the name is disassociated from the STREAM.

2014 **3.232 NaN (Not a Number)**

2015 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-  
 2016 point numbers. Not all systems support NaN values.

2017 **3.233 Native Language**

2018 A computer user's spoken or written language, such as American English, British English,  
 2019 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2020 **3.234 Negative Response**

2021 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
 2022 keyword **noexpr**, matching an extended regular expression in the current locale.

2023 **Note:** The *LC\_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 164).

2024 **3.235 Network**

2025 A collection of interconnected hosts.

2026 **Note:** The term "network" in POSIX.1-2008 is used to refer to the network of hosts. The term "batch  
 2027 system" is used to refer to the network of batch servers.

2028 **3.236 Network Address**

2029 A network-visible identifier used to designate specific endpoints in a network. Specific  
 2030 endpoints on host systems have addresses, and host systems may also have addresses.

2031 **3.237 Network Byte Order**

2032 The way of representing any integer type such that, when transmitted over a network via a  
 2033 network endpoint, the **int** type is transmitted as an appropriate number of octets with the most  
 2034 significant octet first, followed by any other octets in descending order of significance.

2035 **Note:** This order is more commonly known as big-endian ordering. See also [Section 4.9](#) (on page 110).

2036 **3.238 Newline Character (<newline>)**

2037 A character that in the output stream indicates that printing should start at the beginning of the  
 2038 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this  
 2039 character is the exact sequence transmitted to an output device by the system to accomplish the  
 2040 movement to the next line.

2041 **3.239 Nice Value**

2042 A number used as advice to the system to alter process scheduling. Numerically smaller values  
 2043 give a process additional preference when scheduling a process to run. Numerically larger  
 2044 values reduce the preference and make a process less likely to run. Typically, a process with a  
 2045 smaller nice value runs to completion more quickly than an equivalent process with a higher  
 2046 nice value. The symbol {NZERO} specifies the default nice value of the system.

2047 **3.240 Non-Blocking**

2048 A property of an open file description that causes function calls involving it to return without  
 2049 delay when it is detected that the requested action associated with the function call cannot be  
 2050 completed without unknown delay.

2051 **Note:** The exact semantics are dependent on the type of file associated with the open file description.  
 2052 For data reads from devices such as ttys and FIFOs, this property causes the read to return  
 2053 immediately when no data was available. Similarly, for writes, it causes the call to return  
 2054 immediately when the thread would otherwise be delayed in the write operation; for example,  
 2055 because no space was available. For networking, it causes functions not to await protocol events  
 2056 (for example, acknowledgements) to occur. See also XSH [Section 2.10.7](#) (on page 519).

2057 **3.241 Non-Spacing Characters**

2058 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001  
 2059 standard coded graphic character set, which is used in combination with other characters to  
 2060 form composite graphic symbols.

2061 **3.242 NUL**

2062 A character with all bits set to zero.

2063 **3.243 Null Byte**

2064 A byte with all bits set to zero.

2065 **3.244 Null Pointer**

2066 A pointer obtained by converting an integer constant expression with the value 0, or such an  
2067 expression cast to type **void \***, to a pointer type; for example, **(char \*)0**. The C language  
2068 guarantees that a null pointer compares unequal to a pointer to any object or function, so it is  
2069 used by many functions that return pointers to indicate an error.

2070 **3.245 Null String**2071 See *Empty String* in [Section 3.146](#) (on page 56).2072 **3.246 Null Wide-Character Code**

2073 A wide-character code with all bits set to zero.

2074 **3.247 Number-Sign Character (<number-sign>)**

2075 The character ' # ', also known as hash sign.

2076 **3.248 Object File**

2077 A regular file containing the output of a compiler, formatted as input to a linkage editor for  
2078 linking with other object files into an executable form. The methods of linking are unspecified  
2079 and may involve the dynamic linking of objects at runtime. The internal format of an object file  
2080 is unspecified, but a conforming application cannot assume an object file is a text file.

2081 **3.249 Octet**

2082 Unit of data representation that consists of eight contiguous bits.

2083 **3.250 Offset Maximum**

2084 An attribute of an open file description representing the largest value that can be used as a file  
2085 offset.

2086 **3.251 Opaque Address**

2087 An address such that the entity making use of it requires no details about its contents or format.

2088 **3.252 Open File**

2089 A file that is currently associated with a file descriptor.

2090 **3.253 Open File Description**

2091 A record of how a process or group of processes is accessing a file. Each file descriptor refers to  
2092 exactly one open file description, but an open file description can be referred to by more than  
2093 one file descriptor. The file offset, file status, and file access modes are attributes of an open file  
2094 description.

2095 **3.254 Operand**

2096 An argument to a command that is generally used as an object supplying information to a utility  
2097 necessary to complete its processing. Operands generally follow the options in a command line.

2098 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2099 **3.255 Operator**

2100 In the shell command language, either a control operator or a redirection operator.

2101 **3.256 Option**

2102 An argument to a command that is generally used to specify changes in the utility's default  
2103 behavior.

2104 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2105 **3.257 Option-Argument**

2106 A parameter that follows certain options. In some cases an option-argument is included within  
2107 the same argument string as the option—in most cases it is the next argument.

2108 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2109 **3.258 Orientation**

2110 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2111 **Note:** For further information, see XSH [Section 2.5.2](#) (on page 493).

2112 **3.259 Orphaned Process Group**

2113 A process group in which the parent of every member is either itself a member of the group or is  
2114 not a member of the group's session.

2115 **3.260 Page**

2116 The granularity of process memory mapping or locking.

2117 Physical memory and memory objects can be mapped into the address space of a process on  
2118 page boundaries and in integral multiples of pages. Process address space can be locked into  
2119 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2120 **3.261 Page Size**

2121 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On  
2122 systems that have segment rather than page-based memory architectures, the term "page"  
2123 means a segment.

2124 **3.262 Parameter**

2125 In the shell command language, an entity that stores values. There are three types of parameters:  
 2126 variables (named parameters), positional parameters, and special parameters. Parameter  
 2127 expansion is accomplished by introducing a parameter with the '\$' character.

2128 **Note:** See also XCU [Section 2.5](#) (on page 2301).

2129 In the C language, an object declared as part of a function declaration or definition that acquires  
 2130 a value on entry to the function, or an identifier following the macro name in a function-like  
 2131 macro definition.

2132 **3.263 Parent Directory**

2133 When discussing a given directory, the directory that both contains a directory entry for the  
 2134 given directory and is represented by the pathname dot-dot in the given directory.

2135 When discussing other types of files, a directory containing a directory entry for the file under  
 2136 discussion.

2137 This concept does not apply to dot and dot-dot.

2138 **3.264 Parent Process**

2139 The process which created (or inherited) the process under discussion.

2140 **3.265 Parent Process ID**

2141 An attribute of a new process identifying the parent of the process. The parent process ID of a  
 2142 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime  
 2143 has ended, the parent process ID is the process ID of an implementation-defined system process.

2144 **3.266 Pathname**

2145 A character string that is used to identify a file. In the context of POSIX.1-2008, a pathname may  
 2146 be limited to {PATH\_MAX} bytes, including the terminating null byte. It has an optional  
 2147 beginning <slash>, followed by zero or more filenames separated by <slash> characters. A  
 2148 pathname may optionally contain one or more trailing <slash> characters. Multiple successive  
 2149 <slash> characters are considered to be the same as one <slash>, except for the case of exactly  
 2150 two leading <slash> characters.

2151 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).

2152 **3.267 Pathname Component**2153 See *Filename* in [Section 3.170](#) (on page 60).2154 **3.268 Path Prefix**

2155 The part of a pathname up to, but not including, the last component and any trailing <slash>  
 2156 characters, unless the pathname consists entirely of <slash> characters, in which case the path  
 2157 prefix is '/' for a pathname containing either a single <slash> or three or more <slash>  
 2158 characters, and '//' for the pathname //. The path prefix of a pathname containing no <slash>  
 2159 characters is empty, but is treated as referring to the current working directory.

2160 **Note:** The term is used both in the sense of identifying part of a pathname that forms the prefix and of  
 2161 joining a non-empty path prefix to a filename to form a pathname. In the latter case, the path  
 2162 prefix need not have a trailing <slash> (in which case the joining is done with a <slash>  
 2163 character).

2164 **3.269 Pattern**

2165 A sequence of characters used either with regular expression notation or for pathname  
 2166 expansion, as a means of selecting various character strings or pathnames, respectively.

2167 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

2168 See also XCU [Section 2.6.6](#) (on page 2311).

2169 The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-2008 always  
 2170 indicates the type of pattern being referred to in the immediate context of the use of the term.

2171 **3.270 Period Character (<period>)**

2172 The character '.'. The term "period" is contrasted with dot (see also [Section 3.136](#), on page 54),  
 2173 which is used to describe a specific directory entry.

2174 **3.271 Permissions**

2175 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2176 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 108).

2177 **3.272 Persistence**

2178 A mode for semaphores, shared memory, and message queues requiring that the object and its  
 2179 state (including data, if any) are preserved after the object is no longer referenced by any  
 2180 process.

2181 Persistence of an object does not imply that the state of the object is maintained across a system  
2182 crash or a system reboot.

### 2183 3.273 Pipe

2184 An object identical to a FIFO which has no links in the file hierarchy.

2185 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

### 2186 3.274 Polling

2187 A scheduling scheme whereby the local process periodically checks until the pre-specified  
2188 events (for example, read, write) have occurred.

### 2189 3.275 Portable Character Set

2190 The collection of characters that are required to be present in all locales supported by  
2191 conforming systems.

2192 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

2193 This term is contrasted against the smaller portable filename character set; see also [Section 3.276](#).

### 2194 3.276 Portable Filename Character Set

2195 The set of characters from which portable filenames are constructed.

2196 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
2197 a b c d e f g h i j k l m n o p q r s t u v w x y z  
2198 0 1 2 3 4 5 6 7 8 9 . \_ -

2199 The last three characters are the <period>, <underscore>, and <hyphen> characters, respectively.

2200 **3.277 Positional Parameter**

2201 In the shell command language, a parameter denoted by a single digit or one or more digits in  
2202 curly braces.

2203 **Note:** For further information, see XCU [Section 2.5.1](#) (on page 2301).

2204 **3.278 Preallocation**

2205 The reservation of resources in a system for a particular use.

2206 Preallocation does not imply that the resources are immediately allocated to that use, but merely  
2207 indicates that they are guaranteed to be available in bounded time when needed.

2208 **3.279 Preempted Process (or Thread)**

2209 A running thread whose execution is suspended due to another thread becoming runnable at a  
2210 higher priority.

2211 **3.280 Previous Job**

2212 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the  
2213 current job exits. There is at most one previous job; see also [Section 3.203](#) (on page 65).

2214 **3.281 Printable Character**

2215 One of the characters included in the **print** character classification of the *LC\_CTYPE* category in  
2216 the current locale.

2217 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

2218 **3.282 Printable File**

2219 A text file consisting only of the characters included in the **print** and **space** character  
2220 classifications of the *LC\_CTYPE* category and the <backspace>, all in the current locale.

2221 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

2222 **3.283 Priority**

2223 A non-negative integer associated with processes or threads whose value is constrained to a  
 2224 range defined by the applicable scheduling policy. Numerically higher values represent higher  
 2225 priorities.

2226 **3.284 Priority Band**

2227 The queuing order applied to normal priority STREAMS messages. High priority STREAMS  
 2228 messages are not grouped by priority bands. The only differentiation made by the STREAMS  
 2229 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate  
 2230 between priority bands.

2231 **3.285 Priority Inversion**

2232 A condition in which a thread that is not voluntarily suspended (waiting for an event or time  
 2233 delay) is not running while a lower priority thread is running. Such blocking of the higher  
 2234 priority thread is often caused by contention for a shared resource.

2235 **3.286 Priority Scheduling**

2236 A performance and determinism improvement facility to allow applications to determine the  
 2237 order in which threads that are ready to run are granted access to processor resources.

2238 **3.287 Priority-Based Scheduling**

2239 Scheduling in which the selection of a running thread is determined by the priorities of the  
 2240 runnable processes or threads.

2241 **3.288 Privilege**

2242 See *Appropriate Privileges* in [Section 3.20](#) (on page 36).

2243 **3.289 Process**

2244 An address space with one or more threads executing within that address space, and the  
2245 required system resources for those threads.

2246 **Note:** Many of the system resources defined by POSIX.1-2008 are shared among all of the threads  
2247 within a process. These include the process ID, the parent process ID, process group ID, session  
2248 membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID,  
2249 supplementary group IDs, current working directory, root directory, file mode creation mask  
2250 and file descriptors.

2251 **3.290 Process Group**

2252 A collection of processes that permits the signaling of related processes. Each process in the  
2253 system is a member of a process group that is identified by a process group ID. A newly created  
2254 process joins the process group of its creator.

2255 **3.291 Process Group ID**

2256 The unique positive integer identifier representing a process group during its lifetime.

2257 **Note:** See also *Process Group ID Reuse* defined in [Section 4.13](#) (on page 112).

2258 **3.292 Process Group Leader**

2259 A process whose process ID is the same as its process group ID.

2260 **3.293 Process Group Lifetime**

2261 The period of time that begins when a process group is created and ends when the last  
2262 remaining process in the group leaves the group, due either to the end of the lifetime of the last  
2263 process or to the last remaining process calling the *setsid()* or *setpgid()* functions.

2264 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of  
2265 POSIX.1-2008.

2266 **3.294 Process ID**

2267 The unique positive integer identifier representing a process during its lifetime.

2268 **Note:** See also *Process ID Reuse* defined in [Section 4.13](#) (on page 112).2269 **3.295 Process Lifetime**

2270 The period of time that begins when a process is created and ends when its process ID is  
 2271 returned to the system. After a process is created by *fork()*, *posix\_spawn()*, or *posix\_spawnp()*, it is  
 2272 considered active. At least one thread of control and address space exist until it terminates. It  
 2273 then enters an inactive state where certain resources may be returned to the system, although  
 2274 some resources, such as the process ID, are still in use. When another process executes a *wait()*,  
 2275 *waitid()*, or *waitpid()* function for an inactive process, the remaining resources are returned to  
 2276 the system. The last resource to be returned to the system is the process ID. At this time, the  
 2277 lifetime of the process ends.

2278 **Note:** The *fork()*, *posix\_spawn()*, *posix\_spawnp()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in  
 2279 detail in the System Interfaces volume of POSIX.1-2008.

2280 **3.296 Process Memory Locking**

2281 A performance improvement facility to bind application programs into the high-performance  
 2282 random access memory of a computer system. This avoids potential latencies introduced by the  
 2283 operating system in storing parts of a program that were not recently referenced on secondary  
 2284 memory devices.

2285 **3.297 Process Termination**

2286 There are two kinds of process termination:

2287 1. Normal termination occurs by a return from *main()*, when requested with the *exit()*,  
 2288 *\_exit()*, or *\_Exit()* functions; or when the last thread in the process terminates by  
 2289 returning from its start function, by calling the *pthread\_exit()* function, or through  
 2290 cancellation.

2291 2. Abnormal termination occurs when requested by the *abort()* function or when some  
 2292 signals are received.

2293 **Note:** The *\_exit()*, *\_Exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces  
 2294 volume of POSIX.1-2008.

2295 **3.298 Process-To-Process Communication**

2296 The transfer of data between processes.

2297 **3.299 Process Virtual Time**

2298 The measurement of time in units elapsed by the system clock while a process is executing.

2299 **3.300 Program**2300 A prepared sequence of instructions to the system to accomplish a defined task. The term  
2301 “program” in POSIX.1-2008 encompasses applications written in the Shell Command Language,  
2302 complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.2303 **3.301 Protocol**

2304 A set of semantic and syntactic rules for exchanging information.

2305 **3.302 Pseudo-Terminal**2306 A facility that provides an interface that is identical to the terminal subsystem, except where  
2307 noted otherwise in POSIX.1-2008. A pseudo-terminal is composed of two devices: the “master  
2308 device” and a “slave device”. The slave device provides processes with an interface that is  
2309 identical to the terminal interface, although there need not be hardware behind that interface.  
2310 Anything written on the master device is presented to the slave as an input and anything  
2311 written on the slave device is presented as an input on the master side.2312 **3.303 Radix Character**

2313 The character that separates the integer part of a number from the fractional part.

2314 **3.304 Read-Only File System**

2315 A file system that has implementation-defined characteristics restricting modifications.

2316 **Note:** File Times Update is described in detail in [Section 4.8](#) (on page 109).2317 **3.305 Read-Write Lock**2318 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
2319 read-only access to data while allowing only one thread to have write access at any given time.  
2320 They are typically used to protect data that is read-only more frequently than it is changed.2321 Read-write locks can be used to synchronize threads in the current process and other processes if  
2322 they are allocated in memory that is writable and shared among the cooperating processes and  
2323 have been initialized for this behavior.

2324 **3.306 Real Group ID**

2325 The attribute of a process that, at the time of process creation, identifies the group of the user  
2326 who created the process; see also [Section 3.188](#) (on page 63).

2327 **3.307 Real Time**

2328 Time measured as total units elapsed by the system clock without regard to which thread is  
2329 executing.

2330 **3.308 Realtime Signal Extension**

2331 A determinism improvement facility to enable asynchronous signal notifications to an  
2332 application to be queued without impacting compatibility with the existing signal functions.

2333 **3.309 Real User ID**

2334 The attribute of a process that, at the time of process creation, identifies the user who created the  
2335 process; see also [Section 3.428](#) (on page 102).

2336 **3.310 Record**

2337 A collection of related data units or words which is treated as a unit.

2338 **3.311 Redirection**

2339 In the shell command language, a method of associating files with the input or output of  
2340 commands.

2341 **Note:** For further information, see XCU [Section 2.7](#) (on page 2312).

2342 **3.312 Redirection Operator**

2343 In the shell command language, a token that performs a redirection function. It is one of the  
 2344 following symbols:

2345 < > >| << >> <& >& <<- >>

2346 **3.313 Referenced Shared Memory Object**

2347 A shared memory object that is open or has one or more mappings defined on it.

2348 **3.314 Refresh**

2349 To ensure that the information on the user's terminal screen is up-to-date.

2350 **3.315 Regular Expression**

2351 A pattern that selects specific strings from a set of character strings.

2352 **Note:** Regular Expressions are described in detail in [Chapter 9](#) (on page 181).

2353 **3.316 Region**

2354 In the context of the address space of a process, a sequence of addresses.

2355 In the context of a file, a sequence of offsets.

2356 **3.317 Regular File**

2357 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the  
 2358 system.

2359 **3.318 Relative Pathname**

2360 A pathname not beginning with a &lt;slash&gt; character.

2361 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).2362 **3.319 Relocatable File**2363 A file holding code or data suitable for linking with other object files to create an executable or a  
2364 shared object file.2365 **3.320 Relocation**2366 The process of connecting symbolic references with symbolic definitions. For example, when a  
2367 program calls a function, the associated call instruction transfers control to the proper  
2368 destination address at execution.2369 **3.321 Requested Batch Service**2370 A service that is either rejected or performed prior to a response from the service to the  
2371 requester.2372 **3.322 (Time) Resolution**

2373 The minimum time interval that a clock can measure or whose passage a timer can detect.

2374 **3.323 Robust Mutex**2375 A mutex with the *robust* attribute set.2376 **Note:** The *robust* attribute is defined in detail by the `pthread_mutexattr_getrobust()` function.2377 **3.324 Root Directory**2378 A directory, associated with a process, that is used in pathname resolution for pathnames that  
2379 begin with a <slash> character.2380 **3.325 Runnable Process (or Thread)**

2381 A thread that is capable of being a running thread, but for which no processor is available.

2382 **3.326 Running Process (or Thread)**

2383 A thread currently executing on a processor. On multi-processor systems there may be more  
2384 than one such thread in a system at a time.

2385 **3.327 Saved Resource Limits**

2386 An attribute of a process that provides some flexibility in the handling of unrepresentable  
2387 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2388 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of  
2389 POSIX.1-2008.

2390 **3.328 Saved Set-Group-ID**

2391 An attribute of a process that allows some flexibility in the assignment of the effective group ID  
2392 attribute, as described in the *exec* family of functions and *setgid()*.

2393 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of  
2394 POSIX.1-2008.

2395 **3.329 Saved Set-User-ID**

2396 An attribute of a process that allows some flexibility in the assignment of the effective user ID  
2397 attribute, as described in the *exec* family of functions and *setuid()*.

2398 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of  
2399 POSIX.1-2008.

2400 **3.330 Scheduling**

2401 The application of a policy to select a runnable process or thread to become a running process or  
2402 thread, or to alter one or more of the thread lists.

2403 **3.331 Scheduling Allocation Domain**

2404 The set of processors on which an individual thread can be scheduled at any given time.

2405 **3.332 Scheduling Contention Scope**

2406 A property of a thread that defines the set of threads against which that thread competes for  
2407 resources.

2408 For example, in a scheduling decision, threads sharing scheduling contention scope compete for  
 2409 processor resources. In POSIX.1-2008, a thread has scheduling contention scope of either  
 2410 PTHREAD\_SCOPE\_SYSTEM or PTHREAD\_SCOPE\_PROCESS.

### 2411 3.333 Scheduling Policy

2412 A set of rules that is used to determine the order of execution of processes or threads to achieve  
 2413 some goal.

2414 **Note:** Scheduling Policy is defined in detail in [Section 4.14](#) (on page 112).

### 2415 3.334 Screen

2416 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a  
 2417 physical display device or may occupy the entire physical area of the display device.

### 2418 3.335 Scroll

2419 To move the representation of data vertically or horizontally relative to the terminal screen.  
 2420 There are two types of scrolling:

- 2421 1. The cursor moves with the data.
- 2422 2. The cursor remains stationary while the data moves.

### 2423 3.336 Semaphore

2424 A minimum synchronization primitive to serve as a basis for more complex synchronization  
 2425 mechanisms to be defined by the application program.

2426 **Note:** Semaphores are defined in detail in [Section 4.16](#) (on page 113).

2427 **3.337 Session**

2428 A collection of process groups established for job control purposes. Each process group is a  
 2429 member of a session. A process is considered to be a member of the session of which its process  
 2430 group is a member. A newly created process joins the session of its creator. A process can alter  
 2431 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2432 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

2433 **3.338 Session Leader**

2434 A process that has created a session.

2435 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of  
 2436 POSIX.1-2008.

2437 **3.339 Session Lifetime**

2438 The period between when a session is created and the end of the lifetime of all the process  
 2439 groups that remain as members of the session.

2440 **3.340 Shared Memory Object**

2441 An object that represents memory that can be mapped concurrently into the address space of  
 2442 more than one process.

2443 **3.341 Shell**

2444 A program that interprets sequences of text input as commands. It may operate on an input  
 2445 stream or it may interactively prompt and read commands from a terminal.

2446 **3.342 Shell, the**

2447 The Shell Command Language Interpreter; a specific instance of a shell.

2448 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of  
 2449 POSIX.1-2008.

**2450 3.343 Shell Script**

2451 A file containing shell commands. If the file is made executable, it can be executed by specifying  
2452 its name as a simple command. Execution of a shell script causes a shell to execute the  
2453 commands within the script. Alternatively, a shell can be requested to execute the commands in  
2454 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2455 **Note:** Simple Commands are defined in detail in XCU Section 2.9.1 (on page 2316).

2456 The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

**2457 3.344 Signal**

2458 A mechanism by which a process or thread may be notified of, or affected by, an event occurring  
2459 in the system. Examples of such events include hardware exceptions and specific actions by  
2460 processes. The term signal is also used to refer to the event itself.

**2461 3.345 Signal Stack**

2462 Memory established for a thread, in which signal handlers catching signals sent to that thread  
2463 are executed.

**2464 3.346 Single-Quote Character**

2465 The character designated by ' \ ' in the C language, also known as <apostrophe>.

**2466 3.347 Slash Character (<slash>)**

2467 The character ' / ', also known as solidus.

**2468 3.348 Socket**

2469 A file of a particular type that is used as a communications endpoint for process-to-process  
2470 communication as described in the System Interfaces volume of POSIX.1-2008.

**2471 3.349 Socket Address**

2472 An address associated with a socket or remote endpoint, including an address family identifier  
2473 and addressing information specific to that address family. The address may include multiple  
2474 parts, such as a network address associated with a host system and an identifier for a specific  
2475 endpoint.

2476 **3.350 Soft Limit**

2477 A resource limitation established for each process that the process may set to any value less than  
2478 or equal to the hard limit.

2479 **3.351 Source Code**

2480 When dealing with the Shell Command Language, input to the command language interpreter.  
2481 The term “shell script” is synonymous with this meaning.

2482 When dealing with an ISO/IEC-conforming programming language, source code is input to a  
2483 compiler conforming to that ISO/IEC standard.

2484 Source code also refers to the input statements prepared for the following standard utilities: *awk*,  
2485 *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

2486 Source code can also refer to a collection of sources meeting any or all of these meanings.

2487 **Note:** The *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and  
2488 Utilities volume of POSIX.1-2008.

2489 **3.352 Space Character (<space>)**

2490 The character defined in the portable character set as <space>. The <space> character is a  
2491 member of the **space** character class of the current locale, but represents the single character, and  
2492 not all of the possible members of the class; see also [Section 3.434](#) (on page 103).

2493 **3.353 Spawn**

2494 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient  
2495 replacement for *fork()/exec*.

2496 **3.354 Special Built-In**

2497 See *Built-In Utility* in [Section 3.83](#) (on page 46).

2498 **3.355 Special Parameter**

2499 In the shell command language, a parameter named by a single character from the following list:

2500 \* @ # ? ! - \$ 0

2501 **Note:** For further information, see XCU [Section 2.5.2](#) (on page 2302).2502 **3.356 Spin Lock**

2503 A synchronization object used to allow multiple threads to serialize their access to shared data.

2504 **3.357 Sporadic Server**2505 A scheduling policy for threads and processes that reserves a certain amount of execution  
2506 capacity for processing aperiodic events at a given priority level.2507 **3.358 Standard Error**

2508 An output stream usually intended to be used for diagnostic messages.

2509 **3.359 Standard Input**

2510 An input stream usually intended to be used for primary data input.

2511 **3.360 Standard Output**

2512 An output stream usually intended to be used for primary data output.

2513 **3.361 Standard Utilities**

2514 The utilities described in the Shell and Utilities volume of POSIX.1-2008.

2515 **3.362 Stream**

2516           Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence  
 2517           of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,  
 2518           *fmemopen()*, *fopen()*, *open\_memstream()*, or *popen()* functions, and are associated with a file  
 2519           descriptor. A stream provides the additional services of user-selectable buffering and formatted  
 2520           input and output; see also [Section 3.363](#).

2521           **Note:**       For further information, see XSH [Section 2.5](#) (on page 490).

2522                       The *fdopen()*, *fmemopen()*, *fopen()*, *open\_memstream()*, and *popen()* functions are defined in detail  
 2523                       in the System Interfaces volume of POSIX.1-2008.

2524 **3.363 STREAM**

2525           Appearing in uppercase, STREAM refers to a full-duplex connection between a process and an  
 2526           open device or pseudo-device. It optionally includes one or more intermediate processing  
 2527           modules that are interposed between the process end of the STREAM and the device driver (or  
 2528           pseudo-device driver) end of the STREAM; see also [Section 3.362](#).

2529           **Note:**       For further information, see XSH [Section 2.6](#) (on page 494).

2530 **3.364 STREAM End**

2531           The STREAM end is the driver end of the STREAM and is also known as the downstream end of  
 2532           the STREAM.

2533 **3.365 STREAM Head**

2534           The STREAM head is the beginning of the STREAM and is at the boundary between the system  
 2535           and the application process. This is also known as the upstream end of the STREAM.

2536 **3.366 STREAMS Multiplexor**

2537           A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected  
 2538           above is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected  
 2539           below is referred to as 1-to-N or “lower multiplexing”.

2540 **3.367 String**

2541           A contiguous sequence of bytes terminated by and including the first null byte.

2542 **3.368 Subshell**

2543 A shell execution environment, distinguished from the main or current shell execution  
2544 environment.

2545 **Note:** For further information, see XCU [Section 2.12](#) (on page 2331).

2546 **3.369 Successfully Transferred**

2547 For a write operation to a regular file, when the system ensures that all data written is readable  
2548 on any subsequent open of the file (even one that follows a system or power failure) in the  
2549 absence of a failure of the physical storage medium.

2550 For a read operation, when an image of the data on the physical storage medium is available to  
2551 the requesting process.

2552 **3.370 Supplementary Group ID**

2553 An attribute of a process used in determining file access permissions. A process has up to  
2554 {NGROUPS\_MAX} supplementary group IDs in addition to the effective group ID. The  
2555 supplementary group IDs of a process are set to the supplementary group IDs of the parent  
2556 process when the process is created.

2557 **3.371 Suspended Job**

2558 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the  
2559 process group to stop. A suspended job is a background job, but a background job is not  
2560 necessarily a suspended job.

2561 **3.372 Symbolic Constant**

2562 An object-like macro defined with a constant value.

2563 Unless stated otherwise, the following shall apply to every symbolic constant:

- 2564 • It expands to a compile-time constant expression with an integer type.
- 2565 • It may be defined as another type of constant—e.g., an enumeration constant—as well as  
2566 being a macro.
- 2567 • It need not be usable in **#if** preprocessing directives.

2568 **3.373 Symbolic Link**

2569 A type of file with the property that when the file is encountered during pathname resolution, a  
 2570 string stored by the file is used to modify the pathname resolution. The stored string has a  
 2571 length of {SYMLINK\_MAX} bytes or fewer.

2572 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 111).

2573 **3.374 Synchronized Input and Output**

2574 A determinism and robustness improvement mechanism to enhance the data input and output  
 2575 mechanisms, so that an application can ensure that the data being manipulated is physically  
 2576 present on secondary mass storage devices.

2577 **3.375 Synchronized I/O Completion**

2578 The state of an I/O operation that has either been successfully transferred or diagnosed as  
 2579 unsuccessful.

2580 **3.376 Synchronized I/O Data Integrity Completion**

2581 For read, when the operation has been completed or diagnosed if unsuccessful. The read is  
 2582 complete only when an image of the data has been successfully transferred to the requesting  
 2583 process. If there were any pending write requests affecting the data to be read at the time that  
 2584 the synchronized read operation was requested, these write requests are successfully transferred  
 2585 prior to reading the data.

2586 For write, when the operation has been completed or diagnosed if unsuccessful. The write is  
 2587 complete only when the data specified in the write request is successfully transferred and all file  
 2588 system information required to retrieve the data is successfully transferred.

2589 File attributes that are not necessary for data retrieval (access time, modification time, status  
 2590 change time) need not be successfully transferred prior to returning to the calling process.

2591 **3.377 Synchronized I/O File Integrity Completion**

2592 Identical to a synchronized I/O data integrity completion with the addition that all file  
 2593 attributes relative to the I/O operation (including access time, modification time, status change  
 2594 time) are successfully transferred prior to returning to the calling process.

2595 **3.378 Synchronized I/O Operation**

2596 An I/O operation performed on a file that provides the application assurance of the integrity of  
 2597 its data and files.

### 2598 3.379 Synchronous I/O Operation

2599 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the  
2600 processor until that I/O operation completes.

2601 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or  
2602 synchronized I/O file integrity completion.

### 2603 3.380 Synchronously-Generated Signal

2604 A signal that is attributable to a specific thread.

2605 For example, a thread executing an illegal instruction or touching invalid memory causes a  
2606 synchronously-generated signal. Being synchronous is a property of how the signal was  
2607 generated and not a property of the signal number.

### 2608 3.381 System

2609 An implementation of POSIX.1-2008.

### 2610 3.382 System Boot

2611 An unspecified sequence of events that may result in the loss of transitory data; that is, data that  
2612 is not saved in permanent storage. For example, message queues, shared memory, semaphores,  
2613 and processes.

### 2614 3.383 System Clock

2615 A clock with at least one second resolution that contains seconds since the Epoch.

### 2616 3.384 System Console

2617 A device that receives messages sent by the *syslog()* function, and the *fntmsg()* function when  
2618 the MM\_CONSOLE flag is set.

2619 **Note:** The *syslog()* and *fntmsg()* functions are defined in detail in the System Interfaces volume of  
2620 POSIX.1-2008.

### 2621 3.385 System Crash

2622 An interval initiated by an unspecified circumstance that causes all processes (possibly other  
2623 than special system processes) to be terminated in an undefined manner, after which any

2624 changes to the state and contents of files created or written to by an application prior to the  
2625 interval are undefined, except as required elsewhere in POSIX.1-2008.

### 2626 **3.386 System Databases**

2627 An implementation provides two system databases: the “group database” (see also [Section](#)  
2628 [3.187](#), on page 63) and the “user database” (see also [Section 3.427](#), on page 101).

### 2629 **3.387 System Documentation**

2630 All documentation provided with an implementation except for the conformance document.  
2631 Electronically distributed documents for an implementation are considered part of the system  
2632 documentation.

### 2633 **3.388 System Process**

2634 An object other than a process executing an application, that is provided by the system and has a  
2635 process ID.

### 2636 **3.389 System Reboot**

2637 See *System Boot* defined in [Section 3.382](#) (on page 95).

### 2638 **3.390 System Trace Event**

2639 A trace event that is generated by the implementation, in response either to a system-initiated  
2640 action or to an application-requested action, except for a call to *posix\_trace\_event()*. When  
2641 supported by the implementation, a system-initiated action generates a process-independent  
2642 system trace event and an application-requested action generates a process-dependent system  
2643 trace event. For a system trace event not defined by POSIX.1-2008, the associated trace event  
2644 type identifier is derived from the implementation-defined name for this trace event, and the  
2645 associated data is of implementation-defined content and length.

### 2646 **3.391 System-Wide**

2647 Pertaining to events occurring in all processes existing in an implementation at a given point in  
2648 time.

2649 **3.392 Tab Character (<tab>)**

2650 A character that in the output stream indicates that printing or displaying should start at the  
 2651 next horizontal tabulation position on the current line. It is the character designated by '`\t`' in  
 2652 the C language. If the current position is at or past the last defined horizontal tabulation  
 2653 position, the behavior is unspecified. It is unspecified whether this character is the exact  
 2654 sequence transmitted to an output device by the system to accomplish the tabulation.

2655 **3.393 Terminal (or Terminal Device)**

2656 A character special file that obeys the specifications of the general terminal interface.

2657 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

2658 **3.394 Text Column**

2659 A roughly rectangular block of characters capable of being laid out side-by-side next to other  
 2660 text columns on an output page or terminal screen. The widths of text columns are measured in  
 2661 column positions.

2662 **3.395 Text File**

2663 A file that contains characters organized into zero or more lines. The lines do not contain NUL  
 2664 characters and none can exceed {`LINE_MAX`} bytes in length, including the <newline>  
 2665 character. Although POSIX.1-2008 does not distinguish between text files and binary files (see  
 2666 the ISO C standard), many utilities only produce predictable or meaningful output when  
 2667 operating on text files. The standard utilities that have such restrictions always specify "text  
 2668 files" in their STDIN or INPUT FILES sections.

2669 **3.396 Thread**

2670 A single flow of control within a process. Each thread has its own thread ID, scheduling priority  
 2671 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to  
 2672 support a flow of control. Anything whose address may be determined by a thread, including  
 2673 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage  
 2674 obtained through implementation-defined functions, and automatic variables, are accessible to  
 2675 all threads in the same process.

2676 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

2677 **3.397 Thread ID**

2678 Each thread in a process is uniquely identified during its lifetime by a value of type `pthread_t`  
 2679 called a thread ID.

2680 **3.398 Thread List**

2681 An ordered set of runnable threads that all have the same ordinal value for their priority.

2682 The ordering of threads on the list is determined by a scheduling policy or policies. The set of  
2683 thread lists includes all runnable threads in the system.2684 **3.399 Thread-Safe**2685 A function that may be safely invoked concurrently by multiple threads. Each function defined  
2686 in the System Interfaces volume of POSIX.1-2008 is thread-safe unless explicitly stated  
2687 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is  
2688 accessing static storage, or objects shared among threads.2689 **3.400 Thread-Specific Data Key**2690 A process global handle of type `pthread_key_t` which is used for naming thread-specific data.2691 Although the same key value may be used by different threads, the values bound to the key by  
2692 `pthread_setspecific()` and accessed by `pthread_getspecific()` are maintained on a per-thread basis  
2693 and persist for the life of the calling thread.2694 **Note:** The `pthread_getspecific()` and `pthread_setspecific()` functions are defined in detail in the System  
2695 Interfaces volume of POSIX.1-2008.2696 **3.401 Tilde Character (<tilde>)**

2697 The character '~'.

2698 **3.402 Timeouts**2699 A method of limiting the length of time an interface will block; see also [Section 3.76](#) (on page 44).2700 **3.403 Timer**2701 A mechanism that can notify a thread when the time as measured by a particular clock has  
2702 reached or passed a specified value, or when a specified amount of time has passed.2703 **3.404 Timer Overrun**2704 A condition that occurs each time a timer, for which there is already an expiration signal queued  
2705 to the process, expires.

2706 **3.405 Token**

2707 In the shell command language, a sequence of characters that the shell considers as a single unit  
2708 when reading input. A token is either an operator or a word.

2709 **Note:** The rules for reading input are defined in detail in XCU [Section 2.3](#) (on page 2299).

2710 **3.406 Trace Analyzer Process**

2711 A process that extracts trace events from a trace stream to retrieve information about the  
2712 behavior of an application.

2713 **3.407 Trace Controller Process**

2714 A process that creates a trace stream for tracing a process.

2715 **3.408 Trace Event**

2716 A data object that represents an action executed by the system, and that is recorded in a trace  
2717 stream.

2718 **3.409 Trace Event Type**

2719 A data object type that defines a class of trace event.

2720 **3.410 Trace Event Type Mapping**

2721 A one-to-one mapping between trace event types and trace event names.

2722 **3.411 Trace Filter**

2723 A filter that allows the trace controller process to specify those trace event types that are to be  
2724 ignored; that is, not generated.

2725 **3.412 Trace Generation Version**

2726 A data object that is an implementation-defined character string, generated by the trace system  
2727 and describing the origin and version of the trace system.

2728 **3.413 Trace Log**

2729 The flushed image of a trace stream, if the trace stream is created with a trace log.

2730 **3.414 Trace Point**

2731 An action that may cause a trace event to be generated.

2732 **3.415 Trace Stream**2733 An opaque object that contains trace events plus internal data needed to interpret those trace  
2734 events.2735 **3.416 Trace Stream Identifier**

2736 A handle to manage tracing operations in a trace stream.

2737 **3.417 Trace System**2738 A system that allows both system and user trace events to be generated into a trace stream.  
2739 These trace events can be retrieved later.2740 **3.418 Traced Process**2741 A process for which at least one trace stream has been created. A traced process is also called a  
2742 target process.2743 **3.419 Tracing Status of a Trace Stream**2744 A status that describes the state of an active trace stream. The tracing status of a trace stream can  
2745 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:  
2746 running or suspended.2747 **3.420 Typed Memory Name Space**2748 A system-wide name space that contains the names of the typed memory objects present in the  
2749 system. It is configurable for a given implementation.

2750 **3.421 Typed Memory Object**

2751 A combination of a typed memory pool and a typed memory port. The entire contents of the  
 2752 pool are accessible from the port. The typed memory object is identified through a name that  
 2753 belongs to the typed memory name space.

2754 **3.422 Typed Memory Pool**

2755 An extent of memory with the same operational characteristics. Typed memory pools may be  
 2756 contained within each other.

2757 **3.423 Typed Memory Port**

2758 A hardware access path to one or more typed memory pools.

2759 **3.424 Unbind**

2760 Remove the association between a network address and an endpoint.

2761 **3.425 Unit Data**

2762 See *Datagram* in [Section 3.124](#) (on page 53).

2763 **3.426 Upshifting**

2764 The conversion of a lowercase character that has a single-character uppercase representation into  
 2765 this uppercase representation.

2766 **3.427 User Database**

2767 A system database that contains at least the following information for each user ID:

- 2768 • User name
- 2769 • Numerical user ID
- 2770 • Initial numerical group ID
- 2771 • Initial working directory
- 2772 • Initial user program

2773 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under  
 2774 which the initial values are operative are implementation-defined.

- 2775 If the initial user program field is null, an implementation-defined program is used.
- 2776 If the initial working directory field is null, the interpretation of that field is implementation-defined.
- 2777
- 2778 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

### 2779 3.428 User ID

- 2780 A non-negative integer that is used to identify a system user. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a saved set-user-ID.
- 2781
- 2782

### 2783 3.429 User Name

- 2784 A string that is used to identify a user; see also [Section 3.427](#) (on page 101). To be portable across systems conforming to POSIX.1-2008, the value is composed of characters from the portable filename character set. The <hyphen> character should not be used as the first character of a portable user name.
- 2785
- 2786
- 2787

### 2788 3.430 User Trace Event

- 2789 A trace event that is generated explicitly by the application as a result of a call to *posix\_trace\_event()*.
- 2790

### 2791 3.431 Utility

- 2792 A program, excluding special built-in utilities provided as part of the Shell Command Language, that can be called by name from a shell to perform a specific task, or related set of tasks.
- 2793
- 2794 **Note:** For further information on special built-in utilities, see XCU [Section 2.14](#) (on page 2334).

2795 **3.432 Variable**

2796 In the shell command language, a named parameter.

2797 **Note:** For further information, see XCU [Section 2.5](#) (on page 2301).2798 **3.433 Vertical-Tab Character (<vertical-tab>)**

2799 A character that in the output stream indicates that printing should start at the next vertical  
 2800 tabulation position. It is the character designated by '`\v`' in the C language. If the current  
 2801 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is  
 2802 unspecified whether this character is the exact sequence transmitted to an output device by the  
 2803 system to accomplish the tabulation.

2804 **3.434 White Space**

2805 A sequence of one or more characters that belong to the **space** character class as defined via the  
 2806 `LC_CTYPE` category in the current locale.

2807 In the POSIX locale, white space consists of one or more `<blank>` (`<space>` and `<tab>`  
 2808 characters), `<newline>`, `<carriage-return>`, `<form-feed>`, and `<vertical-tab>` characters.

2809 **3.435 Wide-Character Code (C Language)**

2810 An integer value corresponding to a single graphic symbol or control code.

2811 **Note:** C Language Wide-Character Codes are defined in detail in [Section 6.3](#) (on page 129).2812 **3.436 Wide-Character Input/Output Functions**

2813 The functions that perform wide-oriented input from streams or wide-oriented output to  
 2814 streams: `fgetwc()`, `fgetwts()`, `fputwc()`, `fputwts()`, `fwprintf()`, `fwscanf()`, `getwc()`, `getwchar()`, `putwc()`,  
 2815 `putwchar()`, `ungetwc()`, `vwprintf()`, `vwscanf()`, `vwprintf()`, `vwscanf()`, `wprintf()`, and `wscanf()`.

2816 **Note:** These functions are defined in detail in the System Interfaces volume of POSIX.1-2008.2817 **3.437 Wide-Character String**

2818 A contiguous sequence of wide-character codes terminated by and including the first null wide-  
 2819 character code.

2820 **3.438 Word**

2821 In the shell command language, a token other than an operator. In some cases a word is also a  
2822 portion of a word token: in the various forms of parameter expansion, such as  $\{name-word\}$ ,  
2823 and variable assignment, such as  $name=word$ , the word is the portion of the token depicted by  
2824 *word*. The concept of a word is no longer applicable following word expansions—only fields  
2825 remain.

2826 **Note:** For further information, see XCU [Section 2.6.2](#) (on page 2306) and [Section 2.6](#) (on page 2305).

2827 **3.439 Working Directory (or Current Working Directory)**

2828 A directory, associated with a process, that is used in pathname resolution for pathnames that do  
2829 not begin with a <slash> character.

2830 **3.440 Worldwide Portability Interface**

2831 Functions for handling characters in a codeset-independent manner.

2832 **3.441 Write**

2833 To output characters to a file, such as standard output or standard error. Unless otherwise stated,  
2834 standard output is the default output destination for all uses of the term “write”; see the  
2835 distinction between display and write in [Section 3.133](#) (on page 54).

2836 **3.442 XSI**

2837 The X/Open System Interfaces (XSI) option is the core application programming interface for C  
2838 and *sh* programming for systems conforming to the Single UNIX Specification. This is a  
2839 superset of the mandatory requirements for conformance to POSIX.1-2008.

2840 **3.443 XSI-Conformant**

2841 A system which allows an application to be built using a set of services that are consistent across  
2842 all systems that conform to POSIX.1-2008 and that support the XSI option.

2843 **Note:** See also [Chapter 2](#) (on page 15).

2844 **3.444 Zombie Process**

2845 A process that has terminated and that is deleted when its exit status has been reported to  
2846 another process which is waiting for that process to terminate.

2847 **3.445  $\pm 0$** 

2848 The algebraic sign provides additional information about any variable that has the value zero  
2849 when the representation allows the sign to be determined.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

2850

Chapter 4

2851

## General Concepts

2852

For the purposes of POSIX.1-2008, the general concepts given in [Chapter 4](#) apply.

2853

2854

2855

**Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

2856

### 4.1 Concurrent Execution

2857

2858

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

2859

### 4.2 Directory Protection

2860

2861

If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

2862

2863

2864

2865

2866

2867

- The effective user ID of the process is the same as that of the owner ID of the file.
- The effective user ID of the process is the same as that of the owner ID of the directory.
- The process has appropriate privileges.
- Optionally, the file is writable by the process. Whether or not files that are writable by the process can be removed or renamed is implementation-defined.

If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

2868

### 4.3 Extended Security Controls

2869

2870

2871

2872

An implementation may provide implementation-defined extended security controls (see [Section 3.160](#), on page 58). These permit an implementation to provide security mechanisms to implement different security policies than those described in POSIX.1-2008. These mechanisms shall not alter or override the defined semantics of any of the interfaces in POSIX.1-2008.

#### 2873 4.4 File Access Permissions

2874 The standard file access control mechanism uses the file permission bits, as described below.

2875 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An  
2876 additional access control mechanism shall only further restrict the access permissions defined by  
2877 the file permission bits. An alternate file access control mechanism shall:

- 2878 • Specify file permission bits for the file owner class, file group class, and file other class of  
2879 that file, corresponding to the access permissions.
- 2880 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with  
2881 appropriate privileges.
- 2882 • Be disabled for a file after the file permission bits are changed for that file with *chmod()*.  
2883 The disabling of the alternate mechanism need not disable any additional mechanisms  
2884 supported by an implementation.

2885 Whenever a process requests file access permission for read, write, or execute/search, if no  
2886 additional mechanism denies access, access shall be determined as follows:

- 2887 • If a process has appropriate privileges:
  - 2888 — If read, write, or directory search permission is requested, access shall be granted.
  - 2889 — If execute permission is requested, access shall be granted if execute permission is  
2890 granted to at least one user by the file permission bits or by an alternate access  
2891 control mechanism; otherwise, access shall be denied.
- 2892 • Otherwise:
  - 2893 — The file permission bits of a file contain read, write, and execute/search permissions  
2894 for the file owner class, file group class, and file other class.
  - 2895 — Access shall be granted if an alternate access control mechanism is not enabled and  
2896 the requested access permission bit is set for the class (file owner class, file group  
2897 class, or file other class) to which the process belongs, or if an alternate access control  
2898 mechanism is enabled and it allows the requested access; otherwise, access shall be  
2899 denied.

2900 POSIX.1-2008 does not provide a way to open a directory for searching. It is unspecified  
2901 whether directory search permission is granted based on the file access modes of the directory's  
2902 file descriptor or on the mode of the directory at the time the directory is searched.

#### 2903 4.5 File Hierarchy

2904 Files in the system are organized in a hierarchical structure in which all of the non-terminal  
2905 nodes are directories and all of the terminal nodes are any other type of file. Since multiple  
2906 directory entries may refer to the same file, the hierarchy is properly described as a “directed  
2907 graph”.

## 2908 4.6 Filenames

2909 Uppercase and lowercase letters shall retain their unique identities between conforming  
2910 implementations.

## 2911 4.7 Filename Portability

2912 For a filename to be portable across implementations conforming to POSIX.1-2008, it shall  
2913 consist only of the portable filename character set as defined in Section 3.276 (on page 77).

2914 Portable filenames shall not have the <hyphen> character as the first character since this may  
2915 cause problems when filenames are passed as command line arguments.

## 2916 4.8 File Times Update

2917 Each file has three distinct associated timestamps: the time of last data access, the time of last  
2918 data modification, and the time the file status last changed. These values are returned in the file  
2919 characteristics structure **struct stat**, as described in <sys/stat.h> (on page 388).

2920 Each function or utility in POSIX.1-2008 that reads or writes data (even if the data does not  
2921 change) or performs an operation to change file status (even if the file status does not change)  
2922 indicates which of the appropriate timestamps shall be marked for update. If an implementation  
2923 of such a function or utility marks for update one of these timestamps in a place or time not  
2924 specified by POSIX.1-2008, this shall be documented, except that any changes caused by  
2925 pathname resolution need not be documented. For the other functions or utilities in  
2926 POSIX.1-2008 (those that are not explicitly required to read or write file data or change file  
2927 status, but that in some implementations happen to do so), the effect is unspecified.

2928 An implementation may update timestamps that are marked for update immediately, or it may  
2929 update such timestamps periodically. At the point in time when an update occurs, any marked  
2930 timestamps shall be set to the current time and the update marks shall be cleared. All  
2931 timestamps that are marked for update shall be updated when the file ceases to be open by any  
2932 process or before a *fstat()*, *fstatat()*, *fsync()*, *futimens()*, *lstat()*, *stat()*, *utime()*, *utimensat()*, or  
2933 *utimes()* is successfully performed on the file. Other times at which updates are done are  
2934 unspecified. Marks for update, and updates themselves, shall not be done for files on read-only  
2935 file systems; see Section 3.304 (on page 82).

2936 The resolution of timestamps of files in a file system is implementation-defined, but shall be no  
2937 coarser than one-second resolution. The three timestamps shall always have values that are  
2938 supported by the file system. Whenever any of a file's timestamps are to be set to a value *V*  
2939 according to the rules of the preceding paragraphs of this section, the implementation shall  
2940 immediately set the timestamp to the greatest value supported by the file system that is not  
2941 greater than *V*.

## 2942 4.9 Host and Network Byte Orders

2943 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned  
2944 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be  
2945 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,  
2946 and with the first (lowest-addressed) octet holding the most-significant bits. This is called  
2947 “network byte order”.

2948 Network byte order may not be convenient for processing actual values. For this, it is more  
2949 sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host  
2950 byte order:

- 2951 • The most significant bit might not be stored in the first byte in address order.
- 2952 • Bits might not be allocated to bytes in any obvious order at all.

2953 8-bit values stored in `uint8_t` objects do not require conversion to or from host byte order, as  
2954 they have the same representation. 16 and 32-bit values can be converted using the `htonl()`,  
2955 `htons()`, `ntohl()`, and `ntohs()` functions. When reading data that is to be converted to host byte  
2956 order, it should either be received directly into a `uint16_t` or `uint32_t` object or should be copied  
2957 from an array of bytes using `memcpy()` or similar. Passing the data through other types could  
2958 cause the byte order to be changed. Similar considerations apply when sending data.

## 2959 4.10 Measurement of Execution Time

2960 The mechanism used to measure execution time shall be implementation-defined. The  
2961 implementation shall also define to whom the CPU time that is consumed by interrupt handlers  
2962 and system services on behalf of the operating system will be charged. See [Section 3.118](#) (on  
2963 page 52).

## 2964 4.11 Memory Synchronization

2965 Applications shall ensure that access to any memory location by more than one thread of control  
2966 (threads or processes) is restricted such that no thread of control can read or modify a memory  
2967 location while another thread of control may be modifying it. Such access is restricted using  
2968 functions that synchronize thread execution and also synchronize memory with respect to other  
2969 threads. The following functions synchronize memory with respect to other threads:

2970 <code>fork()</code>	<code>pthread_mutex_trylock()</code>	<code>pthread_rwlock_unlock()</code>
2971 <code>pthread_barrier_wait()</code>	<code>pthread_mutex_unlock()</code>	<code>pthread_rwlock_wrlock()</code>
2972 <code>pthread_cond_broadcast()</code>	<code>pthread_spin_lock()</code>	<code>sem_post()</code>
2973 <code>pthread_cond_signal()</code>	<code>pthread_spin_trylock()</code>	<code>sem_timedwait()</code>
2974 <code>pthread_cond_timedwait()</code>	<code>pthread_spin_unlock()</code>	<code>sem_trywait()</code>
2975 <code>pthread_cond_wait()</code>	<code>pthread_rwlock_rdlock()</code>	<code>sem_wait()</code>
2976 <code>pthread_create()</code>	<code>pthread_rwlock_timedrdlock()</code>	<code>semctl()</code>
2977 <code>pthread_join()</code>	<code>pthread_rwlock_timedwrlock()</code>	<code>semop()</code>
2978 <code>pthread_mutex_lock()</code>	<code>pthread_rwlock_tryrdlock()</code>	<code>wait()</code>
2979 <code>pthread_mutex_timedlock()</code>	<code>pthread_rwlock_trywrlock()</code>	<code>waitpid()</code>

2980 The `pthread_once()` function shall synchronize memory for the first call in each thread for a given  
2981 `pthread_once_t` object.

2982 The `pthread_mutex_lock()` function need not synchronize memory if the mutex type if  
 2983 PTHREAD\_MUTEX\_RECURSIVE and the calling thread already owns the mutex. The  
 2984 `pthread_mutex_unlock()` function need not synchronize memory if the mutex type is  
 2985 PTHREAD\_MUTEX\_RECURSIVE and the mutex has a lock count greater than one.

2986 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified  
 2987 whether the invocation causes memory to be synchronized.

2988 Applications may allow more than one thread of control to read a memory location  
 2989 simultaneously.

## 2990 4.12 Pathname Resolution

2991 Pathname resolution is performed for a process to resolve a pathname to a particular directory  
 2992 entry for a file in the file hierarchy. There may be multiple pathnames that resolve to the same  
 2993 directory entry, and multiple directory entries for the same file. When a process resolves a  
 2994 pathname of an existing directory entry, the entire pathname shall be resolved as described  
 2995 below. When a process resolves a pathname of a directory entry that is to be created immediately  
 2996 after the pathname is resolved, pathname resolution terminates when all components of the path  
 2997 prefix of the last component have been resolved. It is then the responsibility of the process to  
 2998 create the final component.

2999 Each filename in the pathname is located in the directory specified by its predecessor (for  
 3000 example, in the pathname fragment `a/b`, file `b` is located in directory `a`). Pathname resolution  
 3001 shall fail if this cannot be accomplished. If the pathname begins with a `<slash>`, the predecessor  
 3002 of the first filename in the pathname shall be taken to be the root directory of the process (such  
 3003 pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a  
 3004 `<slash>`, the predecessor of the first filename of the pathname shall be taken to be either the  
 3005 current working directory of the process or for certain interfaces the directory identified by a file  
 3006 descriptor passed to the interface (such pathnames are referred to as “relative pathnames”).

3007 The interpretation of a pathname component is dependent on the value of `{NAME_MAX}` and  
 3008 `_POSIX_NO_TRUNC` associated with the path prefix of that component. If any pathname  
 3009 component is longer than `{NAME_MAX}`, the implementation shall consider this an error.

3010 A pathname that contains at least one non-`<slash>` character and that ends with one or more  
 3011 trailing `<slash>` characters shall not be resolved successfully unless the last pathname  
 3012 component before the trailing `<slash>` characters names an existing directory or a directory  
 3013 entry that is to be created for a directory immediately after the pathname is resolved. Interfaces  
 3014 using pathname resolution may specify additional constraints<sup>6</sup> when a pathname that does not  
 3015 name an existing directory contains at least one non-`<slash>` character and contains one or more  
 3016 trailing `<slash>` characters.

3017 If a symbolic link is encountered during pathname resolution, the behavior shall depend on  
 3018 whether the pathname component is at the end of the pathname and on the function being  
 3019 performed. If all of the following are true, then pathname resolution is complete:

- 3020 1. This is the last pathname component of the pathname.
- 3021 2. The pathname has no trailing `<slash>`.

---

3022 6. The only interfaces that further constrain pathnames in POSIX.1-2008 are the `rename()` and `renameat()` functions (see XSH `rename()`)  
 3023 and the `mv` utility (see XCU `mv`).

3024 3. The function is required to act on the symbolic link itself, or certain arguments direct that  
3025 the function act on the symbolic link itself.

3026 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the  
3027 symbolic link. If the combined length exceeds {PATH\_MAX}, and the implementation considers  
3028 this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be  
3029 returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.  
3030 If the resulting pathname does not begin with a <slash>, the predecessor of the first filename of  
3031 the pathname is taken to be the directory containing the symbolic link.

3032 If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and  
3033 return an error indication. The same may happen if during the resolution process more symbolic  
3034 links were followed than the implementation allows. This implementation-defined limit shall  
3035 not be smaller than {SYMLOOP\_MAX}.

3036 The special filename dot shall refer to the directory specified by its predecessor. The special  
3037 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,  
3038 in the root directory, dot-dot may refer to the root directory itself.

3039 A pathname consisting of a single <slash> shall resolve to the root directory of the process. A  
3040 null pathname shall not be successfully resolved. A pathname that begins with two successive  
3041 <slash> characters may be interpreted in an implementation-defined manner, although more  
3042 than two leading <slash> characters shall be treated as a single <slash> character.

3043 Pathname resolution for a given pathname shall yield the same results when used by any  
3044 interface in POSIX.1-2008 as long as there are no changes to any files evaluated during pathname  
3045 resolution for the given pathname between resolutions.

### 3046 4.13 Process ID Reuse

3047 A process group ID shall not be reused by the system until the process group lifetime ends.

3048 A process ID shall not be reused by the system until the process lifetime ends. In addition, if  
3049 there exists a process group whose process group ID is equal to that process ID, the process ID  
3050 shall not be reused by the system until the process group lifetime ends. A process that is not a  
3051 system process shall not have a process ID of 1.

### 3052 4.14 Scheduling Policy

3053 A scheduling policy affects process or thread ordering:

- 3054 • When a process or thread is a running thread and it becomes a blocked thread
- 3055 • When a process or thread is a running thread and it becomes a preempted thread
- 3056 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3057 • When a running thread calls a function that can change the priority or scheduling policy of  
3058 a process or thread
- 3059 • In other scheduling policy-defined circumstances

3060 Conforming implementations shall define the manner in which each of the scheduling policies  
3061 may modify the priorities or otherwise affect the ordering of processes or threads at each of the  
3062 occurrences listed above. Additionally, conforming implementations shall define in what other

3063 circumstances and in what manner each scheduling policy may modify the priorities or affect  
3064 the ordering of processes or threads.

#### 3065 4.15 Seconds Since the Epoch

3066 A value that approximates the number of seconds that have elapsed since the Epoch. A  
3067 Coordinated Universal Time name (specified in terms of seconds (*tm\_sec*), minutes (*tm\_min*),  
3068 hours (*tm\_hour*), days since January 1 of the year (*tm\_yday*), and calendar year minus 1900  
3069 (*tm\_year*)) is related to a time represented as seconds since the Epoch, according to the  
3070 expression below.

3071 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and  
3072 the value is non-negative, the value is related to a Coordinated Universal Time name according  
3073 to the C-language expression, where *tm\_sec*, *tm\_min*, *tm\_hour*, *tm\_yday*, and *tm\_year* are all  
3074 integer types:

$$3075 \quad tm\_sec + tm\_min*60 + tm\_hour*3600 + tm\_yday*86400 +$$

$$3076 \quad (tm\_year-70)*31536000 + ((tm\_year-69)/4)*86400 -$$

$$3077 \quad ((tm\_year-1)/100)*86400 + ((tm\_year+299)/400)*86400$$

3078 The relationship between the actual time of day and the current value for seconds since the  
3079 Epoch is unspecified.

3080 How any changes to the value of seconds since the Epoch are made to align to a desired  
3081 relationship with the current actual time is implementation-defined. As represented in seconds  
3082 since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3083 **Note:** The last three terms of the expression add in a day for each year that follows a leap year starting  
3084 with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973,  
3085 the second subtracts a day back out every 100 years starting in 2001, and the third adds a day  
3086 back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that  
3087 is, the remainder is discarded leaving only the integer quotient.

#### 3088 4.16 Semaphore

3089 A minimum synchronization primitive to serve as a basis for more complex synchronization  
3090 mechanisms to be defined by the application program.

3091 For the semaphores associated with the Semaphores option, a semaphore is represented as a  
3092 shareable resource that has a non-negative integer value. When the value is zero, there is a  
3093 (possibly empty) set of threads awaiting the availability of the semaphore.

3094 For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is  
3095 a positive integer (0 through 32767). The *semget()* function can be called to create a set or array of  
3096 semaphores. A semaphore set can contain one or more semaphores up to an implementation-  
3097 defined value.

3098 **Semaphore Lock Operation**

3099 An operation that is applied to a semaphore. If, prior to the operation, the value of the  
 3100 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and  
 3101 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

3102 **Semaphore Unlock Operation**

3103 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in  
 3104 the set of threads awaiting the semaphore, then some thread from that set shall be removed from  
 3105 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

3106 **4.17 Thread-Safety**

3107 Refer to XSH [Section 2.9](#) (on page 507).

3108 **4.18 Tracing**

3109 The trace system allows a traced process to have a selection of events created for it. Traces  
 3110 consist of streams of trace event types.

3111 A trace event type is identified on the one hand by a trace event type name, also referenced as a  
 3112 trace event name, and on the other hand by a trace event type identifier. A trace event name is a  
 3113 human-readable string. A trace event type identifier is an opaque identifier used by the trace  
 3114 system. There shall be a one-to-one relationship between trace event type identifiers and trace  
 3115 event names for a given trace stream and also for a given traced process. The trace event type  
 3116 identifier shall be generated automatically from a trace event name by the trace system either  
 3117 when a trace controller process invokes `posix_trace_trid_eventid_open()` or when an instrumented  
 3118 application process invokes `posix_trace_eventid_open()`. Trace event type identifiers are used to  
 3119 filter trace event types, to allow interpretation of user data, and to identify the kind of trace  
 3120 point that generated a trace event.

3121 Each trace event shall be of a particular trace event type, and associated with a trace event type  
 3122 identifier. The execution of a trace point shall generate a trace event if a trace stream has been  
 3123 created and started for the process that executed the trace point and if the corresponding trace  
 3124 event type identifier is not ignored by filtering.

3125 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a  
 3126 trace log is associated with the trace stream, except that:

- 3127 • For a trace stream, if no resources are available for the event, the event is lost.
- 3128 • For a trace log, if no resources are available for the event, or a flush operation does not  
 3129 succeed, the event is lost.

3130 A trace event recorded in an active trace stream may be retrieved by an application having  
 3131 appropriate privileges.

3132 A trace event recorded in a trace log may be retrieved by an application having appropriate  
 3133 privileges after opening the trace log as a pre-recorded trace stream, with the function  
 3134 `posix_trace_open()`.

3135 When a trace event is reported it is possible to retrieve the following:

- 3136 • A trace event type identifier
- 3137 • A timestamp
- 3138 • The process ID of the traced process, if the trace event is process-dependent
- 3139 • Any optional trace event data including its length
- 3140 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
- 3141 • The program address at which the trace point was invoked

3142 Trace events may be mapped from trace event types to trace event names. One such mapping  
 3143 shall be associated with each trace stream. An active trace stream is associated with a traced  
 3144 process, and also with its children if the Trace Inherit option is supported and also the  
 3145 inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a  
 3146 mapping of the trace event names to trace event type identifiers that have been defined for that  
 3147 process.

3148 Traces can be recorded into either trace streams or trace logs.

3149 The implementation and format of a trace stream are unspecified. A trace stream need not be  
 3150 and generally is not persistent. A trace stream may be either active or pre-recorded:

- 3151 • An active trace stream is a trace stream that has been created and has not yet been shut  
 3152 down. It can be of one of the two following classes:
  - 3153 1. An active trace stream without a trace log that was created with the  
 3154 `posix_trace_create()` function
  - 3155 2. If the Trace Log option is supported, an active trace stream with a trace log that was  
 3156 created with the `posix_trace_create_withlog()` function
- 3157 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using  
 3158 the `posix_trace_open()` function.

3159 An active trace stream can loop. This behavior means that when the resources allocated by the  
 3160 trace system for the trace stream are exhausted, the trace system reuses the resources associated  
 3161 with the oldest recorded trace events to record new trace events.

3162 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This  
 3163 operation causes the trace system to write trace events from the trace stream to the associated  
 3164 trace log, following the defined policies or using an explicit function call. After this operation,  
 3165 the trace system may reuse the resources associated with the flushed trace events.

3166 An active trace stream with or without a trace log can be cleared. This operation shall cause all  
 3167 the resources associated with this trace stream to be reinitialized. The trace stream shall behave  
 3168 as if it was returning from its creation, except that the mapping of trace event type identifiers to  
 3169 trace event names shall not be cleared. If a trace log was associated with this trace stream, the  
 3170 trace log shall also be reinitialized.

3171 A trace log shall be recorded when the `posix_trace_shutdown()` operation is invoked or during  
 3172 tracing, depending on the tracing strategy which is defined by a log policy. After the trace  
 3173 stream has been shut down, the trace information can be retrieved from the associated trace log  
 3174 using the same interface used to retrieve information from an active trace stream.

3175 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance  
 3176 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together  
 3177 with all of its future children. The `posix_pid` member of each trace event in a trace stream shall be  
 3178 the process ID of the traced process.

3179 Each trace point may be an implementation-defined action such as a context switch, or an  
 3180 application-programmed action such as a call to a specific operating system service (for  
 3181 example, *fork()*) or a call to *posix\_trace\_event()*.

3182 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace  
 3183 events. By default, no trace events are filtered.

3184 The results of the tracing operations can be analyzed and monitored by a trace controller process  
 3185 or a trace analyzer process.

3186 Only the trace controller process has control of the trace stream it has created. The control of the  
 3187 operation of a trace stream is done using its corresponding trace stream identifier. The trace  
 3188 controller process is able to:

- 3189 • Initialize the attributes of a trace stream
- 3190 • Create the trace stream
- 3191 • Start and stop tracing
- 3192 • Know the mapping of the traced process
- 3193 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3194 • Shut the trace stream down

3195 A traced process may also be a trace controller process. Only the trace controller process can  
 3196 control its trace stream(s). A trace stream created by a trace controller process shall be shut down  
 3197 if its controller process terminates or executes another file.

3198 A trace controller process may also be a trace analyzer process. Trace analysis can be done  
 3199 concurrently with the traced process or can be done off-line, in the same or in a different  
 3200 platform.

## 3201 4.19 Treatment of Error Conditions for Mathematical Functions

3202 For all the functions in the `<math.h>` header, an application wishing to check for error situations  
 3203 should set *errno* to 0 and call *feclearexcept*(FE\_ALL\_EXCEPT) before calling the function. On  
 3204 return, if *errno* is non-zero or *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW |  
 3205 FE\_UNDERFLOW) is non-zero, an error has occurred.

3206 The following error conditions are defined for all functions in the `<math.h>` header.

### 3207 4.19.1 Domain Error

3208 A "domain error" shall occur if an input argument is outside the domain over which the  
 3209 mathematical function is defined. The description of each function lists any required domain  
 3210 errors; an implementation may define additional domain errors, provided that such errors are  
 3211 consistent with the mathematical definition of the function.

3212 On a domain error, the function shall return an implementation-defined value; if the integer  
 3213 expression (*math\_errhandling* & MATH\_ERRNO) is non-zero, *errno* shall be set to [EDOM]; if  
 3214 the integer expression (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, the "invalid"  
 3215 floating-point exception shall be raised.

3216 **4.19.2 Pole Error**

3217 A “pole error” occurs if the mathematical result of the function is an exact infinity (for example,  
3218  $\log(0.0)$ ).

3219 On a pole error, the function shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or  
3220 HUGE\_VALL according to the return type, with the same sign as the correct value of the  
3221 function; if the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, *errno* shall  
3222 be set to [ERANGE]; if the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-  
3223 zero, the “divide-by-zero” floating-point exception shall be raised.

3224 **4.19.3 Range Error**

3225 A “range error” shall occur if the finite mathematical result of the function cannot be  
3226 represented in an object of the specified type, due to extreme magnitude.

3227 **4.19.3.1 Result Overflows**

3228 A floating result overflows if the magnitude of the mathematical result is finite but so large that  
3229 the mathematical result cannot be represented without extraordinary roundoff error in an object  
3230 of the specified type. If a floating result overflows and default rounding is in effect, then the  
3231 function shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or HUGE\_VALL  
3232 according to the return type, with the same sign as the correct value of the function; if the integer  
3233 expression (math\_errhandling & MATH\_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if  
3234 the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-zero, the “overflow”  
3235 floating-point exception shall be raised.

3236 **4.19.3.2 Result Underflows**

3237 The result underflows if the magnitude of the mathematical result is so small that the  
3238 mathematical result cannot be represented, without extraordinary roundoff error, in an object of  
3239 the specified type. If the result underflows, the function shall return an implementation-defined  
3240 value whose magnitude is no greater than the smallest normalized positive number in the  
3241 specified type; if the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
3242 whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression  
3243 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, whether the “underflow” floating-point  
3244 exception is raised is implementation-defined.

## 3245 4.20 Treatment of NaN Arguments for the Mathematical Functions

3246 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,  
3247 except where stated otherwise.

3248 If a function with one or more NaN arguments returns a NaN result, the result should be the  
3249 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3250 On implementations that support the IEC 60559:1989 standard floating point, functions with  
3251 signaling NaN argument(s) shall be treated as if the function were called with an argument that  
3252 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3253 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are  
3254 evaluated during the function call.

3255 On implementations that support the IEC 60559:1989 standard floating point, for those  
3256 functions that do not have a documented domain error, the following shall apply:

3257 These functions shall fail if:

3258 Domain Error Any argument is a signaling NaN.

3259 Either, the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero and `errno`  
3260 shall be set to `[EDOM]`, or the integer expression (`math_errhandling &`  
3261 `MATH_ERREXCEPT`) is non-zero and the invalid floating-point exception shall be raised.

## 3262 4.21 Utility

3263 A utility program shall be either an executable file, such as might be produced by a compiler or  
3264 linker system from computer source code, or a file of shell source code, directly interpreted by  
3265 the shell. The program may have been produced by the user, provided by the system  
3266 implementor, or acquired from an independent distributor.

3267 The system may implement certain utilities as shell functions (see XCU Section 2.9.5, on page  
3268 2324) or built-in utilities, but only an application that is aware of the command search order (as  
3269 described in XCU Section 2.9.1.1, on page 2317) or of performance characteristics can discern  
3270 differences between the behavior of such a function or built-in utility and that of an executable  
3271 file.

## 3272 4.22 Variable Assignment

3273 In the shell command language, a word consisting of the following parts:

3274 `varname=value`

3275 When used in a context where assignment is defined to occur and at no other time, the `value`  
3276 (representing a word or field) shall be assigned as the value of the variable denoted by `varname`.

3277 **Note:** For further information, see XCU Section 2.9.1 (on page 2316).

3278 The `varname` and `value` parts shall meet the requirements for a name and a word, respectively,  
3279 except that they are delimited by the embedded unquoted `<equals-sign>`, in addition to other  
3280 delimiters.

## General Concepts

## Variable Assignment

- 3281           **Note:**     Additional delimiters are described in XCU [Section 2.3](#) (on page 2299).
- 3282           When a variable assignment is done, the variable shall be created if it did not already exist. If
- 3283           *value* is not specified, the variable shall be given a null value.
- 3284           **Note:**     An alternative form of variable assignment:
- 3285                     *symbol=value*
- 3286           (where *symbol* is a valid word delimited by an <equals-sign>, but not a valid name) produces
- 3287           unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value*
- 3288           syntax.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

3289

Chapter 5

3290

## File Format Notation

3291 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility  
 3292 descriptions use a syntax to describe the data organization within the files, when that  
 3293 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces  
 3294 volume of POSIX.1-2008 *printf()* function, as described in this chapter. When used in STDIN or  
 3295 INPUT FILES sections of the utility descriptions, this syntax describes the format that could  
 3296 have been used to write the text to be read, not a format that could be used by the System  
 3297 Interfaces volume of POSIX.1-2008 *scanf()* function to read the input file.

3298 The description of an individual record is as follows:

3299 "<format>", [<arg1>, <arg2>, ..., <argn>]

3300 The *format* is a character string that contains three types of objects defined below:

- 3301 1. *Characters* that are not "escape sequences" or "conversion specifications", as described  
 3302 below, shall be copied to the output.
- 3303 2. *Escape Sequences* represent non-graphic characters.
- 3304 3. *Conversion Specifications* specify the output format of each argument; see below.

3305 The following characters have the following special meaning in the format string:

3306 ' ' (An empty character position.) Represents one or more <blank> characters.

3307 Δ Represents exactly one <space> character.

3308 [Table 5-1](#) lists escape sequences and associated actions on display devices capable of the action.

3309 **Table 5-1** Escape Sequences and Associated Actions

3310	3311	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325
Escape Sequence	Represents Character	Terminal Action													
\\	<backslash>	Print the <backslash> character.													
\a	<alert>	Attempt to alert the user through audible or visible notification.													
\b	<backspace>	Move the printing position to one column before the current position, unless the current position is the start of a line.													
\f	<form-feed>	Move the printing position to the initial printing position of the next logical page.													
\n	<newline>	Move the printing position to the start of the next line.													
\r	<carriage-return>	Move the printing position to the start of the current line.													
\t	<tab>	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.													
\v	<vertical-tab>	Move the printing position to the start of the next <vertical-tab> position. If there are no more <vertical-tab> positions left on the page, the behavior is undefined.													

3326		Each conversion specification is introduced by the <percent-sign> character ('%'). After the
3327		character '%', the following shall appear in sequence:
3328	<i>flags</i>	Zero or more <i>flags</i> , in any order, that modify the meaning of the conversion
3329		specification.
3330	<i>field width</i>	An optional string of decimal digits to specify a minimum field width. For an
3331		output field, if the converted value has fewer bytes than the field width, it shall be
3332		padded on the left (or right, if the left-adjustment flag ('-'), described below, has
3333		been given) to the field width.
3334	<i>precision</i>	Gives the minimum number of digits to appear for the <i>d</i> , <i>o</i> , <i>i</i> , <i>u</i> , <i>x</i> , or <i>X</i> conversion
3335		specifiers (the field is padded with leading zeros), the number of digits to appear
3336		after the radix character for the <i>e</i> and <i>f</i> conversion specifiers; the maximum
3337		number of significant digits for the <i>g</i> conversion specifier; or the maximum
3338		number of bytes to be written from a string in the <i>s</i> conversion specifier. The
3339		precision shall take the form of a <period> ('.') followed by a decimal digit
3340		string; a null digit string is treated as zero.
3341	<i>conversion specifier characters</i>	
3342		A conversion specifier character (see below) that indicates the type of conversion
3343		to be applied.
3344		The <i>flag</i> characters and their meanings are:
3345	-	The result of the conversion shall be left-justified within the field.
3346	+	The result of a signed conversion shall always begin with a sign ('+' or '-').
3347	<space>	If the first character of a signed conversion is not a sign, a <space> shall be
3348		prefixed to the result. This means that if the <space> and '+' flags both appear,
3349		the <space> flag shall be ignored.
3350	#	The value shall be converted to an alternative form. For <i>c</i> , <i>d</i> , <i>i</i> , <i>u</i> , and <i>s</i>
3351		conversion specifiers, the behavior is undefined. For the <i>o</i> conversion specifier, it
3352		shall increase the precision to force the first digit of the result to be a zero. For <i>x</i> or
3353		<i>X</i> conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively.
3354		For <i>a</i> , <i>A</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , and <i>G</i> conversion specifiers, the result shall always contain a
3355		radix character, even if no digits follow the radix character. For <i>g</i> and <i>G</i> conversion
3356		specifiers, trailing zeros shall not be removed from the result as they usually are.
3357	0	For <i>a</i> , <i>A</i> , <i>d</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , <i>G</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , and <i>X</i> conversion specifiers, leading zeros
3358		(following any indication of sign or base) shall be used to pad to the field width
3359		rather than performing space padding, except when converting an infinity or NaN.
3360		If the '0' and '-' flags both appear, the '0' flag shall be ignored. For <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> ,
3361		<i>x</i> , and <i>X</i> conversion specifiers, if a precision is specified, the '0' flag shall be
3362		ignored. For other conversion specifiers, the behavior is undefined.
3363		Each conversion specifier character shall result in fetching zero or more arguments. The results
3364		are undefined if there are insufficient arguments for the format. If the format is exhausted while
3365		arguments remain, the excess arguments shall be ignored.
3366		The conversion specifiers and their meanings are:
3367	<i>a,A</i>	The floating-point number argument representing a floating-point number shall be
3368		converted in the style "[ - ] 0x <i>h</i> . <i>hhhhp</i> ± <i>d</i> ", where there is one hexadecimal digit
3369		(which shall be non-zero if the argument is a normalized floating-point number
3370		and is otherwise unspecified) before the decimal-point character and the number
3371		of hexadecimal digits after it is equal to the precision; if the precision is missing

## File Format Notation

3372		and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact
3373		representation of the value; if the precision is missing and FLT_RADIX is not a
3374		power of 2, then the precision shall be sufficient to distinguish different floating-
3375		point values in the internal representation used by the utility, except that trailing
3376		zeros may be omitted; if the precision is zero and the # flag is not specified, no
3377		decimal-point character shall appear. The letters "abcdef" shall be used for a
3378		conversion and the letters "ABCDEF" for A conversion. The A conversion specifier
3379		produces a number with X and P instead of x and p. The exponent shall always
3380		contain at least one digit, and only as many more digits as necessary to represent
3381		the decimal exponent of 2. If the value is zero, the exponent shall be zero. A
3382		floating-point number argument representing an infinity or NaN shall be
3383		converted in the style of an f or F conversion specifier.
3384	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal
3385		(o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and
3386		i specifiers shall convert to signed decimal in the style "[-]dddd". The x
3387		conversion specifier shall use the numbers and letters "0123456789abcdef" and
3388		the X conversion specifier shall use the numbers and letters
3389		"0123456789ABCDEF". The <i>precision</i> component of the argument shall specify
3390		the minimum number of digits to appear. If the value being converted can be
3391		represented in fewer digits than the specified minimum, it shall be expanded with
3392		leading zeros. The default precision shall be 1. The result of converting a zero
3393		value with a precision of 0 shall be no characters. If both the field width and
3394		precision are omitted, the implementation may precede, follow, or precede and
3395		follow numeric arguments of types o, i, and u with <blank> characters; arguments
3396		of type o (octal) may be preceded with leading zeros.
3397	f,F	The floating-point number argument shall be written in decimal notation in the
3398		style [-]ddd.ddd, where the number of digits after the radix character (shown here
3399		as a decimal point) shall be equal to the <i>precision</i> specification. The LC_NUMERIC
3400		locale category shall determine the radix character to use in this format. If the
3401		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3402		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3403		A floating-point number argument representing an infinity shall be converted in
3404		one of the styles "[-]inf" or "[-]infinity"; which style is implementation-
3405		defined. A floating-point number argument representing a NaN shall be converted
3406		in one of the styles "[-]nan( <i>n-char-sequence</i> )" or "[-]nan"; which style,
3407		and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F
3408		conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf",
3409		"infinity", or "nan", respectively.
3410	e,E	The floating-point number argument shall be written in the style [-]d.ddde±dd (the
3411		symbol '±' indicates either a <plus-sign> or minus-sign), where there is one digit
3412		before the radix character (shown here as a decimal point) and the number of
3413		digits after it is equal to the precision. The LC_NUMERIC locale category shall
3414		determine the radix character to use in this format. When the precision is missing,
3415		six digits shall be written after the radix character; if the precision is 0, no radix
3416		character shall appear. The E conversion specifier shall produce a number with E
3417		instead of e introducing the exponent. The exponent shall always contain at least
3418		two digits. However, if the value to be written requires an exponent greater than
3419		two digits, additional exponent digits shall be written as necessary.
3420		A floating-point number argument representing an infinity or NaN shall be
3421		converted in the style of an f or F conversion specifier.

3422	<i>g,G</i>	The floating-point number argument shall be written in style <i>f</i> or <i>e</i> (or in style <i>F</i> or <i>E</i> in the case of a <i>G</i> conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style <i>e</i> (or <i>E</i> ) shall be used only if the exponent resulting from the conversion is less than $-4$ or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3423		
3424		
3425		
3426		
3427		
3428		A floating-point number argument representing an infinity or NaN shall be converted in the style of an <i>f</i> or <i>F</i> conversion specifier.
3429		
3430	<i>c</i>	The single-byte character argument shall be written.
3431	<i>s</i>	The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the <i>precision</i> specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.
3432		
3433		
3434		
3435		
3436	<i>%</i>	Write a ' <i>%</i> ' character; no argument is converted.
3437		In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term "field width" should not be confused with the term "precision" used in the description of <i>%s</i> .
3438		
3439		
3440		

#### 3441 Examples

3442	To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where <i>weekday</i> and <i>month</i> are strings:
3443	
3444	<code>"%s,Δ%sΔ%d,Δ%d:%.2d\n" &lt;weekday&gt;, &lt;month&gt;, &lt;day&gt;, &lt;hour&gt;, &lt;min&gt;</code>
3445	To show ' <i>π</i> ' written to 5 decimal places:
3446	<code>"piΔ=Δ%.5f\n", &lt;value of π&gt;</code>
3447	To show an input file format consisting of five <colon>-separated fields:
3448	<code>"%s:%s:%s:%s:%s\n", &lt;arg1&gt;, &lt;arg2&gt;, &lt;arg3&gt;, &lt;arg4&gt;, &lt;arg5&gt;</code>

3449

Chapter 6

3450

## Character Set

### 6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of POSIX.1-2008. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

3457

Table 6-1 Portable Character Set

3458

3459

3460

3461

3462

3463

3464

3465

3466

3467

3468

3469

3470

3471

3472

3473

3474

3475

3476

3477

3478

3479

3480

3481

3482

3483

3484

3485

3486

3487

3488

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	-	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO

	Symbolic Name	Glyph	UCS	Description
3489				
3490	<three>	3	<U0033>	DIGIT THREE
3491	<four>	4	<U0034>	DIGIT FOUR
3492	<five>	5	<U0035>	DIGIT FIVE
3493	<six>	6	<U0036>	DIGIT SIX
3494	<seven>	7	<U0037>	DIGIT SEVEN
3495	<eight>	8	<U0038>	DIGIT EIGHT
3496	<nine>	9	<U0039>	DIGIT NINE
3497	<colon>	:	<U003A>	COLON
3498	<semicolon>	;	<U003B>	SEMICOLON
3499	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3500	<equals-sign>	=	<U003D>	EQUALS SIGN
3501	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3502	<question-mark>	?	<U003F>	QUESTION MARK
3503	<commercial-at>	@	<U0040>	COMMERCIAL AT
3504	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3505	<B>	B	<U0042>	LATIN CAPITAL LETTER B
3506	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3507	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3508	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3509	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3510	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3511	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3512	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3513	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3514	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3515	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3516	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3517	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3518	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3519	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3520	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3521	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3522	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3523	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3524	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3525	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3526	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3527	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3528	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3529	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3530	<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
3531	<backslash>	\	<U005C>	REVERSE SOLIDUS
3532	<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3533	<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
3534	<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
3535	<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3536	<low-line>	_	<U005F>	LOW LINE
3537	<underscore>	_	<U005F>	LOW LINE
3538	<grave-accent>	`	<U0060>	GRAVE ACCENT
3539	<a>	a	<U0061>	LATIN SMALL LETTER A
3540	<b>	b	<U0062>	LATIN SMALL LETTER B

	Symbolic Name	Glyph	UCS	Description
3541				
3542	<c>	c	<U0063>	LATIN SMALL LETTER C
3543	<d>	d	<U0064>	LATIN SMALL LETTER D
3544	<e>	e	<U0065>	LATIN SMALL LETTER E
3545	<f>	f	<U0066>	LATIN SMALL LETTER F
3546	<g>	g	<U0067>	LATIN SMALL LETTER G
3547	<h>	h	<U0068>	LATIN SMALL LETTER H
3548	<i>	i	<U0069>	LATIN SMALL LETTER I
3549	<j>	j	<U006A>	LATIN SMALL LETTER J
3550	<k>	k	<U006B>	LATIN SMALL LETTER K
3551	<l>	l	<U006C>	LATIN SMALL LETTER L
3552	<m>	m	<U006D>	LATIN SMALL LETTER M
3553	<n>	n	<U006E>	LATIN SMALL LETTER N
3554	<o>	o	<U006F>	LATIN SMALL LETTER O
3555	<p>	p	<U0070>	LATIN SMALL LETTER P
3556	<q>	q	<U0071>	LATIN SMALL LETTER Q
3557	<r>	r	<U0072>	LATIN SMALL LETTER R
3558	<s>	s	<U0073>	LATIN SMALL LETTER S
3559	<t>	t	<U0074>	LATIN SMALL LETTER T
3560	<u>	u	<U0075>	LATIN SMALL LETTER U
3561	<v>	v	<U0076>	LATIN SMALL LETTER V
3562	<w>	w	<U0077>	LATIN SMALL LETTER W
3563	<x>	x	<U0078>	LATIN SMALL LETTER X
3564	<y>	y	<U0079>	LATIN SMALL LETTER Y
3565	<z>	z	<U007A>	LATIN SMALL LETTER Z
3566	<left-brace>	{	<U007B>	LEFT CURLY BRACKET
3567	<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
3568	<vertical-line>		<U007C>	VERTICAL LINE
3569	<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
3570	<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
3571	<tilde>	~	<U007E>	TILDE

3572 POSIX.1-2008 uses character names other than the above, but only in an informative way; for  
 3573 example, in examples to illustrate the use of characters beyond the portable character set with  
 3574 the facilities of POSIX.1-2008.

3575 **Table 6-1** (on page 125) defines the characters in the portable character set and the corresponding  
 3576 symbolic character names used to identify each character in a character set description file. The  
 3577 table contains more than one symbolic character name for characters whose traditional name  
 3578 differs from the chosen name. Characters defined in **Table 6-2** (on page 130) may also be used in  
 3579 character set description files.

3580 POSIX.1-2008 places only the following requirements on the encoded values of the characters in  
 3581 the portable character set:

- 3582 • If the encoded values associated with each member of the portable character set are not  
 3583 invariant across all locales supported by the implementation, if an application accesses any  
 3584 pair of locales where the character encodings differ, or accesses data from an application  
 3585 running in a locale which has different encodings from the application's current locale, the  
 3586 results are unspecified.
- 3587 • The encoded values associated with the digits 0 to 9 shall be such that the value of each  
 3588 character after 0 shall be one greater than the value of the previous character.

- 3589
- A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3590
- The encoded values associated with the members of the portable character set are each represented in a single byte. Moreover, if the value is stored in an object of C-language type `char`, it is guaranteed to be positive (except the NUL, which is always zero).
- 3591
- 3592
- 3593 Conforming implementations shall support certain character and character set attributes, as defined in [Section 7.2](#) (on page 136).
- 3594

## 3595 6.2 Character Encoding

3596 The POSIX locale contains the characters in [Table 6-1](#) (on page 125), which have the properties listed in [Section 7.3.1](#) (on page 139). In other locales, the presence, meaning, and representation of any additional characters are locale-specific.

3597

3598

3599 In locales other than the POSIX locale, a character may have a state-dependent encoding. There are two types of these encodings:

3600

- 3601 • A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined if each shift-code and character sequence is considered a multi-byte character. This is done using the concatenated-constant format in a character set description file, as described in [Section 6.4](#) (on page 129). If the implementation supports a character encoding of this type, all of the standard utilities in the Shell and Utilities volume of POSIX.1-2008 shall support it. Use of a single-shift encoding with any of the functions in the System Interfaces volume of POSIX.1-2008 that do not specifically mention the effects of state-dependent encoding is implementation-defined.
- 3602
- 3603
- 3604
- 3605
- 3606
- 3607
- 3608
- 3609 • A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) cannot be defined with the current character set description file format. Use of a locking-shift encoding with any of the standard utilities in the Shell and Utilities volume of POSIX.1-2008 or with any of the functions in the System Interfaces volume of POSIX.1-2008 that do not specifically mention the effects of state-dependent encoding is implementation-defined.
- 3610
- 3611
- 3612
- 3613
- 3614

3615 While in the initial shift state, all characters in the portable character set shall retain their usual interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the sequence shall be a function of the current shift state. A byte with all bits zero shall be interpreted as the null character independent of shift state. Such a byte shall not occur as part of any other character.

3616

3617

3618

3619

3620 The maximum allowable number of bytes in a character in the current locale shall be indicated by `{MB_CUR_MAX}`, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a character set description file; see [Section 6.4](#) (on page 129). The implementation's maximum number of bytes in a character shall be defined by the C-language macro `{MB_LEN_MAX}`.

3621

3622

3623

### 6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's `LC_CTYPE` definition (see Section 7.3.1, on page 139) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see Section 3.246, on page 72), and terminates wide-character strings (see Section 3.435, on page 103). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation. POSIX.1-2008 provides no means of defining a wide-character codeset.

### 6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in POSIX.1-2008 as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

POSIX.1-2008 does not require that multiple character sets or codesets be supported. Although multiple *charmap* files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's `-f` option.

Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in Table 6-1 (on page 125), and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in Table 6-1 (on page 125) shall be included in the file and shall be mapped to a unique coding value, except as noted below. The glyphs represented by the C character constants `'{', '}', '_, '-, '/, '\\", '., and '^'` have more than one symbolic name; all symbolic names for each such glyph shall be included, each with identical encoding. If some or all of the control characters identified in Table 6-2 (on page 130) are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the encodings associated with the symbolic names in Table 6-2 (on page 130) may be the same as characters found in Table 6-1 (on page 125); both names shall be provided for each encoding.

3664

**Table 6-2** Control Character Set

3665	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
3666	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
3667	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
3668	<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
3669	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
3670	<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>

3671 The following declarations can precede the character definitions. Each shall consist of the  
 3672 symbol shown in the following list, starting in column 1, including the surrounding brackets,  
 3673 followed by one or more <blank> characters, followed by the value to be assigned to the symbol.

- 3674 **<code\_set\_name>** The name of the coded character set for which the character set  
 3675 description file is defined. The characters of the name shall be taken from  
 3676 the set of characters with visible glyphs defined in Table 6-1 (on page  
 3677 125).
- 3678 **<mb\_cur\_max>** The maximum number of bytes in a multi-byte character. This shall  
 3679 default to 1.
- 3680 **<mb\_cur\_min>** An unsigned positive integer value that defines the minimum number of  
 3681 XSI bytes in a character for the encoded character set. On XSI-conformant  
 3682 systems, <mb\_cur\_min> shall always be 1.
- 3683 **<escape\_char>** The character used to indicate that the characters following shall be  
 3684 interpreted in a special way, as defined later in this section. This shall  
 3685 default to <backslash> ('\\'), which is the character used in all the  
 3686 following text and examples, unless otherwise noted.
- 3687 **<comment\_char>** The character that, when placed in column 1 of a charmap line, is used to  
 3688 indicate that the line shall be ignored. The default character shall be the  
 3689 <number-sign> ('#').

3690 The character set mapping definitions shall be all the lines immediately following an identifier  
 3691 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line  
 3692 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a  
 3693 <comment\_char> in the first column shall be ignored. Each non-comment line of the character  
 3694 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)  
 3695 shall be in either of two forms:

```
3696 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>
```

3697 or:

```
3698 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,  
3699 <encoding>, <comments>
```

3700 In the first format, the line in the character set mapping definition shall define a single symbolic  
 3701 name and a corresponding encoding. A symbolic name is one or more characters from the set  
 3702 shown with visible glyphs in Table 6-1 (on page 125), enclosed between angle brackets. A  
 3703 character following an escape character is interpreted as itself; for example, the sequence  
 3704 "<\\>>" represents the symbolic name ">" enclosed between angle brackets.

3705 In the second format, the line in the character set mapping definition shall define a range of one  
 3706 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-  
 3707 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 125), followed  
 3708 by an integer formed by one or more decimal digits. Both integers shall contain the same

3709 number of digits. The characters preceding the integer shall be identical in the two symbolic  
 3710 names, and the integer formed by the digits in the second symbolic name shall be equal to or  
 3711 greater than the integer formed by the digits in the first name. This shall be interpreted as a  
 3712 series of symbolic names formed from the common part and each of the integers between the  
 3713 first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the  
 3714 symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3715 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1  
 3716 (on page 125), and shall define the coded character value that corresponds to the character  
 3717 indicated in the table, or the coded character value that corresponds to the control character  
 3718 symbolic name. If the control characters commonly associated with the symbolic names in Table  
 3719 6-2 (on page 130) are supported by the implementation, the symbolic name and the  
 3720 corresponding encoding value shall be included in the file. Additional unique symbolic names  
 3721 may be included. A coded character value can be represented by more than one symbolic name.

3722 The encoding part is expressed as one (for single-byte character values) or more concatenated  
 3723 decimal, octal, or hexadecimal constants in the following formats:

```
3724 "%cd%u", <escape_char>, <decimal byte value>
3725 "%cx%x", <escape_char>, <hexadecimal byte value>
3726 "%co", <escape_char>, <octal byte value>
```

3727 Decimal constants shall be represented by two or three decimal digits, preceded by the escape  
 3728 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".  
 3729 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape  
 3730 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal  
 3731 constants shall be represented by two or three octal digits, preceded by the escape character; for  
 3732 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an  
 3733 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the  
 3734 same type, and interpreted in sequence from first to last with the first byte of the multi-  
 3735 byte character specified by the first byte in the sequence. The manner in which these constants  
 3736 are represented in the character stored in the system is implementation-defined. (This notation  
 3737 was chosen for reasons of portability. There is no requirement that the internal representation in  
 3738 the computer memory be in this same order.) Omitting bytes from a multi-byte character  
 3739 definition produces undefined results.

3740 In lines defining ranges of symbolic names, the encoded value shall be the value for the first  
 3741 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic  
 3742 names defined by the range shall have encoding values in increasing order. Bytes shall be  
 3743 treated as unsigned octets, and carry shall be propagated between the bytes as necessary to  
 3744 represent the range. However, because this causes a null byte in the second or subsequent bytes  
 3745 of a character, such a declaration should not be specified. For example, the line:

```
3746 <j0101>...<j0104> \d129\d254
```

3747 is interpreted as:

```
3748 <j0101> \d129\d254
3749 <j0102> \d129\d255
3750 <j0103> \d130\d00
3751 <j0104> \d130\d01
```

3752 The expanded declaration of the symbol <j0103> in the above example is an invalid  
 3753 specification, because it contains a null byte in the second byte of a character.

3754 The comment is optional.

3755 POSIX.1-2008 provides no means of defining a wide-character codeset.

3756 The following declarations can follow the character set mapping definitions (after the "END  
3757 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in  
3758 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3759 **WIDTH** A non-negative integer value defining the column width (see [Section 3.103](#), on  
3760 page 50) for the printable characters in the coded character set specified in [Table](#)  
3761 [6-1](#) (on page 125) and [Table 6-2](#) (on page 130). Coded character set character values  
3762 shall be defined using symbolic character names followed by column width  
3763 values. Defining a character with more than one **WIDTH** produces undefined  
3764 results. The **END WIDTH** keyword shall be used to terminate the **WIDTH**  
3765 definitions. Specifying the width of a non-printable character in a **WIDTH**  
3766 declaration produces undefined results.

3767 **WIDTH\_DEFAULT**  
3768 A non-negative integer value defining the default column width for any printable  
3769 character not listed by one of the **WIDTH** keywords. If no **WIDTH\_DEFAULT**  
3770 keyword is included in the charmap, the default character width shall be 1.

### 3771 Example

3772 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
3773 WIDTH
3774 <A> 1
3775 <B> 1
3776 <C>...<Z> 1
3777 ...
3778 <fool>...<foon> 2
3779 ...
3780 END WIDTH
```

3781 In this example, the numerical code point values represented by the symbols **<A>** and **<B>** are  
3782 assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on)  
3783 are also assigned a width of 1. Using **<A>...<Z>** would have required fewer lines, but the  
3784 alternative was shown to demonstrate flexibility. The keyword **WIDTH\_DEFAULT** could have  
3785 been added as appropriate.

## 3786 6.4.1 State-Dependent Character Encodings

3787 This section addresses the use of state-dependent character encodings (that is, those in which the  
3788 encoding of a character is dependent on one or more shift codes that may precede it).

3789 A single-shift encoding (where each character not in the initial shift state is preceded by a shift  
3790 code) can be defined in the charmap format if each shift-code/character sequence is considered  
3791 a multi-byte character, defined using the concatenated-constant format described in [Section 6.4](#)  
3792 (on page 129). If the implementation supports a character encoding of this type, all of the  
3793 standard utilities shall support it. A locking-shift encoding (where the state of the character is  
3794 determined by a shift code that may affect more than the single character following it) could be  
3795 defined with an extension to the charmap format described in [Section 6.4](#) (on page 129).

- 3796 If the implementation supports a character encoding of this type, any of the standard utilities  
3797 that describe character (*versus* byte) or text-file manipulation shall have the following  
3798 characteristics:
- 3799 1. The utility shall process the statefully encoded data as a concatenation of state-  
3800 independent characters. The presence of redundant locking shifts shall not affect the  
3801 comparison of two statefully encoded strings.
  - 3802 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall  
3803 produce output that contains locking shifts at the beginning or end of the resulting data,  
3804 if appropriate, to retain correct state information.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

*LC\_CTYPE* Character classification and case conversion.

*LC\_COLLATE* Collation order.

*LC\_MONETARY* Monetary formatting.

*LC\_NUMERIC* Numeric, non-monetary formatting.

*LC\_TIME* Date and time formats.

*LC\_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of POSIX.1-2008 shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of POSIX.1-2008 shall also be modified based on the current locale, as defined by the last call to *setlocale()*.

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the *\_POSIX2\_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of POSIX.1-2008 shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 136). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale (see Section 7.2, on page 136). When the value of a locale environment variable begins with a <slash> ('/'), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is implementation-defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Likewise, if different codesets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the codeset is different from the codeset assumed when the locale was created, the result is also undefined.

Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is

3843 unset or is set to the empty string, the implementation shall set the appropriate environment as  
 3844 defined in [Chapter 8](#) (on page 173).

## 3845 7.2 POSIX Locale

3846 Conforming systems shall provide a POSIX locale, also known as the C locale. The behavior of  
 3847 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the  
 3848 *localedef* utility with input data from the POSIX locale tables in [Section 7.3](#).

3849 The tables in [Section 7.3](#) describe the characteristics and behavior of the POSIX locale for data  
 3850 consisting entirely of characters from the portable character set and the control character set. For  
 3851 other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall  
 3852 be the default locale when the *setlocale()* function is not called.

3853 The POSIX locale can be specified by assigning to the appropriate environment variables the  
 3854 values "C" or "POSIX".

3855 All implementations shall define a locale as the default locale, to be invoked when no  
 3856 environment variables are set, or set to the empty string. This default locale can be the POSIX  
 3857 locale or any other implementation-defined locale. Some implementations may provide facilities  
 3858 for local installation administrators to set the default locale, customizing it for each location.  
 3859 POSIX.1-2008 does not require such a facility.

## 3860 7.3 Locale Definition

3861 The capability to specify additional locales to those provided by an implementation is optional,  
 3862 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only  
 3863 implementation-supplied locales are available. Such locales shall be documented using the  
 3864 format specified in this section.

3865 Locales can be described with the file format presented in this section. The file format is that  
 3866 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the  
 3867 "locale definition file" but no locales shall be affected by this file unless it is processed by  
 3868 *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility  
 3869 shall apply to *localedef* or to any other similar utility used to install locale information using the  
 3870 locale definition file format described here.

3871 The locale definition file shall contain one or more locale category source definitions, and shall  
 3872 not contain more than one definition for the same locale category. If the file contains source  
 3873 definitions for more than one category, implementation-defined categories, if present, shall  
 3874 appear after the categories defined by [Section 7.1](#) (on page 135). A category source definition  
 3875 contains either the definition of a category or a **copy** directive. For a description of the **copy**  
 3876 directive, see *localedef*. In the event that some of the information for a locale category, as  
 3877 specified in this volume of POSIX.1-2008, is missing from the locale source definition, the  
 3878 behavior of that category, if it is referenced, is unspecified.

3879 A category source definition shall consist of a category header, a category body, and a category  
 3880 trailer. A category header shall consist of the character string naming of the category, beginning  
 3881 with the characters `LC_`. The category trailer shall consist of the string "END", followed by one  
 3882 or more <blank> characters and the string used in the corresponding category header.

3883 The category body shall consist of one or more lines of text. Each line shall contain an identifier,  
 3884 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a

3885 particular locale element, or collating elements. In addition to the keywords defined in this  
 3886 volume of POSIX.1-2008, the source can contain implementation-defined keywords. Each  
 3887 keyword within a locale shall have a unique name (that is, two categories cannot have a  
 3888 commonly-named keyword); no keyword shall start with the characters *LC\_*. Identifiers shall be  
 3889 separated from the operands by one or more <blank> characters.

3890 Operands shall be characters, collating elements, or strings of characters. Strings shall be  
 3891 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <escape  
 3892 character>, described below. When a keyword is followed by more than one operand, the  
 3893 operands shall be separated by <semicolon> characters; <blank> characters shall be allowed  
 3894 both before and after a <semicolon>.

3895 The first category header in the file can be preceded by a line modifying the comment character.  
 3896 It shall have the following format, starting in column 1:

```
3897 "comment_char %c\n", <comment character>
```

3898 The comment character shall default to the <number-sign> ('#'). Blank lines and lines  
 3899 containing the <comment character> in the first position shall be ignored.

3900 The first category header in the file can be preceded by a line modifying the escape character to  
 3901 be used in the file. It shall have the following format, starting in column 1:

```
3902 "escape_char %c\n", <escape character>
```

3903 The escape character shall default to <backslash>, which is the character used in all examples  
 3904 shown in this volume of POSIX.1-2008.

3905 A line can be continued by placing an escape character as the last character on the line; this  
 3906 continuation character shall be discarded from the input. Although the implementation need not  
 3907 accept any one portion of a continued line with a length exceeding {LINE\_MAX} bytes, it shall  
 3908 place no limits on the accumulated length of the continued line. Comment lines shall not be  
 3909 continued on a subsequent line using an escaped <newline>.

3910 Individual characters, characters in strings, and collating elements shall be represented using  
 3911 symbolic names, as defined below. In addition, characters can be represented using the  
 3912 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic  
 3913 notation is used, the resultant locale definitions are in many cases not portable between systems.  
 3914 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when  
 3915 used to represent itself it shall be preceded by the escape character. The following rules apply to  
 3916 character representation:

- 3917 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'>  
 3918 and '>'. The symbolic name, including the angle brackets, shall exactly match a  
 3919 symbolic name defined in the charmap file specified via the *localedef* -f option, and it shall  
 3920 be replaced by a character value determined from the value associated with the symbolic  
 3921 name in the charmap file. The use of a symbolic name not found in the charmap file shall  
 3922 constitute an error, unless the category is *LC\_CTYPE* or *LC\_COLLATE*, in which case it  
 3923 shall constitute a warning condition (see *localedef* for a description of actions resulting  
 3924 from errors and warnings). The specification of a symbolic name in a **collating-element**  
 3925 or **collating-symbol** section that duplicates a symbolic name in the charmap file (if  
 3926 present) shall be an error. Use of the escape character or a right angle bracket within a  
 3927 symbolic name is invalid unless the character is preceded by the escape character.

3928 For example:

```
3929 <c>;<c-cedilla> "<M><a><y>"
```

3930 2. A character in the portable character set can be represented by the character itself, in  
 3931 which case the value of the character is implementation-defined. (Implementations may  
 3932 allow other characters to be represented as themselves, but such locale definitions are not  
 3933 portable.) Within a string, the double-quote character, the escape character, and the right  
 3934 angle bracket character shall be escaped (preceded by the escape character) to be  
 3935 interpreted as the character itself. Outside strings, the characters:

3936 `, ; < > escape_char`

3937 shall be escaped to be interpreted as the character itself.

3938 For example:

3939 `c "May"`

3940 3. A character can be represented as an octal constant. An octal constant shall be specified as  
 3941 the escape character followed by two or three octal digits. Each constant shall represent a  
 3942 byte value. Multi-byte values can be represented by concatenated constants specified in  
 3943 byte order with the last constant specifying the least significant byte of the character.

3944 For example:

3945 `\143;\347;\143\150 "\115\141\171"`

3946 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall  
 3947 be specified as the escape character followed by an 'x' followed by two hexadecimal  
 3948 digits. Each constant shall represent a byte value. Multi-byte values can be represented by  
 3949 concatenated constants specified in byte order with the last constant specifying the least  
 3950 significant byte of the character.

3951 For example:

3952 `\x63;\xe7;\x63\x68 "\x4d\x61\x79"`

3953 5. A character can be represented as a decimal constant. A decimal constant shall be  
 3954 specified as the escape character followed by a 'd' followed by two or three decimal  
 3955 digits. Each constant represents a byte value. Multi-byte values can be represented by  
 3956 concatenated constants specified in byte order with the last constant specifying the least  
 3957 significant byte of the character.

3958 For example:

3959 `\d99;\d231;\d99\d104 "\d77\d97\d121"`

3960 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the  
 3961 escape character. Only characters existing in the character set for which the locale definition is  
 3962 created shall be specified, whether using symbolic names, the characters themselves, or octal,  
 3963 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the  
 3964 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not  
 3965 present in the charmap file can be specified and shall be ignored, as specified under item 1  
 3966 above.

3967 **7.3.1 LC\_CTYPE**

3968 The *LC\_CTYPE* category shall define character classification, case conversion, and other  
 3969 character attributes. In addition, a series of characters can be represented by three adjacent  
 3970 <period> characters representing an ellipsis symbol (" . . . "). The ellipsis specification shall be  
 3971 interpreted as meaning that all values between the values preceding and following it represent  
 3972 valid characters. The ellipsis specification shall be valid only within a single encoded character  
 3973 set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as  
 3974 including in the list all characters with an encoded value higher than the encoded value of the  
 3975 character preceding the ellipsis and lower than the encoded value of the character following the  
 3976 ellipsis.

3977 For example:

3978 `\x30; . . . ; \x39;`

3979 includes in the character class all characters with encoded values between the endpoints.

3980 The following keywords shall be recognized. In the descriptions, the term "automatically  
 3981 included" means that it shall not be an error either to include or omit any of the referenced  
 3982 characters; the implementation provides them if missing (even if the entire keyword is missing)  
 3983 and accepts them silently if present. When the implementation automatically includes a missing  
 3984 character, it shall have an encoded value dependent on the charmap file in effect (see the  
 3985 description of the *localedef* -f option); otherwise, it shall have a value derived from an  
 3986 implementation-defined character mapping.

3987 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
 3988 characters. These only need to be specified if the character values (that is, encoding) differ from  
 3989 the implementation default values. It is not possible to define a locale without these  
 3990 automatically included characters unless some implementation extension is used to prevent  
 3991 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and,  
 3992 thus, it might not be possible for conforming applications to work properly.

3993 **copy** Specify the name of an existing locale which shall be used as the definition of  
 3994 this category. If this keyword is specified, no other keyword shall be specified.

3995 **upper** Define characters to be classified as uppercase letters.

3996 In the POSIX locale, the 26 uppercase letters shall be included:

3997 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

3998 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,  
 3999 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as  
 4000 defined in Section 6.4 (on page 129) (the portable character set), are  
 4001 automatically included in this class.

4002 **lower** Define characters to be classified as lowercase letters.

4003 In the POSIX locale, the 26 lowercase letters shall be included:

4004 a b c d e f g h i j k l m n o p q r s t u v w x y z

4005 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,  
 4006 **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the  
 4007 portable character set are automatically included in this class.

4008 **alpha** Define characters to be classified as letters.

4009 In the POSIX locale, all characters in the classes **upper** and **lower** shall be  
 4010 included.

4011		In a locale definition file, no character specified for the keywords <b>cntrl</b> , <b>digit</b> ,
4012		<b>punct</b> , or <b>space</b> shall be specified. Characters classified as either <b>upper</b> or
4013		<b>lower</b> are automatically included in this class.
4014	<b>digit</b>	Define the characters to be classified as numeric digits.
4015		In the POSIX locale, only:
4016		0 1 2 3 4 5 6 7 8 9
4017		shall be included.
4018		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4019		<four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4020		contiguous ascending sequence by numerical value. The digits <zero> to
4021		<nine> of the portable character set are automatically included in this class.
4022	<b>alnum</b>	Define characters to be classified as letters and numeric digits. Only the
4023		characters specified for the <b>alpha</b> and <b>digit</b> keywords shall be specified.
4024		Characters specified for the keywords <b>alpha</b> and <b>digit</b> are automatically
4025		included in this class.
4026	<b>space</b>	Define characters to be classified as white-space characters.
4027		In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
4028		return>, <tab>, and <vertical-tab> shall be included.
4029		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4030		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>graph</b> , or <b>xdigit</b> shall be specified. The <space>, <form-
4031		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4032		character set, and any characters included in the class <b>blank</b> are automatically
4033		included in this class.
4034	<b>cntrl</b>	Define characters to be classified as control characters.
4035		In the POSIX locale, no characters in classes <b>alpha</b> or <b>print</b> shall be included.
4036		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4037		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>punct</b> , <b>graph</b> , <b>print</b> , or <b>xdigit</b> shall be specified.
4038	<b>punct</b>	Define characters to be classified as punctuation characters.
4039		In the POSIX locale, neither the <space> nor any characters in classes <b>alpha</b> ,
4040		<b>digit</b> , or <b>cntrl</b> shall be included.
4041		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4042		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>cntrl</b> , <b>xdigit</b> , or as the <space> shall be specified.
4043	<b>graph</b>	Define characters to be classified as printable characters, not including the
4044		<space>.
4045		In the POSIX locale, all characters in classes <b>alpha</b> , <b>digit</b> , and <b>punct</b> shall be
4046		included; no characters in class <b>cntrl</b> shall be included.
4047		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4048		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , and <b>punct</b> are automatically included in this class. No
4049		character specified for the keyword <b>cntrl</b> shall be specified.
4050	<b>print</b>	Define characters to be classified as printable characters, including the
4051		<space>.
4052		In the POSIX locale, all characters in class <b>graph</b> shall be included; no
4053		characters in class <b>cntrl</b> shall be included.

4054		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4055		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , <b>punct</b> , <b>graph</b> , and the <space> are automatically included
4056		in this class. No character specified for the keyword <b>cntrl</b> shall be specified.
4057	<b>xdigit</b>	Define the characters to be classified as hexadecimal digits.
4058		In the POSIX locale, only:
4059		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4060		shall be included.
4061		In a locale definition file, only the characters defined for the class <b>digit</b> shall be
4062		specified, in contiguous ascending sequence by numerical value, followed by
4063		one or more sets of six characters representing the hexadecimal digits 10 to 15
4064		inclusive, with each set in ascending order (for example, <A>, <B>, <C>, <D>,
4065		<E>, <F>, <a>, <b>, <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
4066		uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
4067		portable character set are automatically included in this class.
4068	<b>blank</b>	Define characters to be classified as <blank> characters.
4069		In the POSIX locale, only the <space> and <tab> shall be included.
4070		In a locale definition file, the <space> and <tab> are automatically included in
4071		this class.
4072	<b>charclass</b>	Define one or more locale-specific character class names as strings separated
4073		by <semicolon> characters. Each named character class can then be defined
4074		subsequently in the <i>LC_CTYPE</i> definition. A character class name shall consist
4075		of at least one and at most {CHARCLASS_NAME_MAX} bytes of
4076		alphanumeric characters from the portable filename character set. The first
4077		character of a character class name shall not be a digit. The name shall not
4078		match any of the <i>LC_CTYPE</i> keywords defined in this volume of
4079		POSIX.1-2008. Future versions of this standard will not specify any <i>LC_CTYPE</i>
4080		keywords containing uppercase letters.
4081	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4082		character class. In the POSIX locale, locale-specific named character classes
4083		need not exist.
4084		If a class name is defined by a <b>charclass</b> keyword, but no characters are
4085		subsequently assigned to it, this is not an error; it represents a class without
4086		any characters belonging to it.
4087		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i>
4088		function, in regular expression and shell pattern-matching bracket
4089		expressions, and by the <i>tr</i> command.
4090	<b>toupper</b>	Define the mapping of lowercase letters to uppercase letters.
4091		In the POSIX locale, at a minimum, the 26 lowercase characters:
4092		a b c d e f g h i j k l m n o p q r s t u v w x y z
4093		shall be mapped to the corresponding 26 uppercase characters:
4094		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4095		In a locale definition file, the operand shall consist of character pairs,
4096		separated by <semicolon> characters. The characters in each character pair
4097		shall be separated by a <comma> and the pair enclosed by parentheses. The

4098 first character in each pair is the lowercase letter, the second the corresponding  
 4099 uppercase letter. Only characters specified for the keywords **lower** and **upper**  
 4100 shall be specified. The lowercase letters <a> to <z>, and their corresponding  
 4101 uppercase letters <A> to <Z>, of the portable character set are automatically  
 4102 included in this mapping, but only when the **toupper** keyword is omitted  
 4103 from the locale definition.

4104 **tolower**

Define the mapping of uppercase letters to lowercase letters.

4105 In the POSIX locale, at a minimum, the 26 uppercase characters:

4106 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

4107 shall be mapped to the corresponding 26 lowercase characters:

4108 a b c d e f g h i j k l m n o p q r s t u v w x y z

4109 In a locale definition file, the operand shall consist of character pairs,  
 4110 separated by <semicolon> characters. The characters in each character pair  
 4111 shall be separated by a <comma> and the pair enclosed by parentheses. The  
 4112 first character in each pair is the uppercase letter, the second the  
 4113 corresponding lowercase letter. Only characters specified for the keywords  
 4114 **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from  
 4115 the locale definition, the mapping is the reverse mapping of the one specified  
 4116 for **toupper**.

4117 The following table shows the character class combinations allowed:

4118 **Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4121 <b>upper</b>	—	—	A	x	x	x	x	A	A	—	x
4122 <b>lower</b>	—	—	A	x	x	x	x	A	A	—	x
4123 <b>alpha</b>	—	—	—	x	x	x	x	A	A	—	x
4124 <b>digit</b>	x	x	x	—	x	x	x	A	A	A	x
4125 <b>space</b>	x	x	x	x	—	*	*	*	*	x	—
4126 <b>cntrl</b>	x	x	x	x	—	—	x	x	x	x	—
4127 <b>punct</b>	x	x	x	x	—	x	—	A	A	x	—
4128 <b>graph</b>	—	—	—	—	—	x	—	—	A	—	—
4129 <b>print</b>	—	—	—	—	—	x	—	—	—	—	—
4130 <b>xdigit</b>	—	—	—	—	x	x	x	A	A	—	x
4131 <b>blank</b>	x	x	x	x	A	—	*	*	*	x	—

4132 **Notes:**

- 4133 1. Explanation of codes:
- 4134 A Automatically included; see text.
- 4135 — Permitted.
- 4136 x Mutually-exclusive.
- 4137 \* See note 2.
- 4138 2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or
- 4139 **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters
- 4140 can be classified as any of **punct**, **graph**, or **print**.

## Locale

## Locale Definition

## 4141 7.3.1.1 LC\_CTYPE Category in the POSIX Locale

4142 The character classifications for the POSIX locale follow; the code listing depicts the *localedef*  
4143 input, and the table represents the same information, sorted by character.

```

4144 LC_CTYPE
4145 # The following is the POSIX locale LC_CTYPE.
4146 # "alpha" is by default "upper" and "lower"
4147 # "alnum" is by definition "alpha" and "digit"
4148 # "print" is by default "alnum", "punct", and the <space>
4149 # "graph" is by default "alnum" and "punct"
4150 #
4151 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4152 <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4153 #
4154 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4155 <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4156 #
4157 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4158 <seven>;<eight>;<nine>
4159 #
4160 space <tab>;<newline>;<vertical-tab>;<form-feed>;\
4161 <carriage-return>;<space>
4162 #
4163 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4164 <form-feed>;<carriage-return>;\
4165 <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4166 <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4167 <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4168 <IS1>;<DEL>
4169 #
4170 punct <exclamation-mark>;<quotation-mark>;<number-sign>;\
4171 <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4172 <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4173 <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4174 <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4175 <greater-than-sign>;<question-mark>;<commercial-at>;\
4176 <left-square-bracket>;<backslash>;<right-square-bracket>;\
4177 <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4178 <vertical-line>;<right-curly-bracket>;<tilde>
4179 #
4180 xdigit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4181 <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4182 #
4183 blank <space>;<tab>
4184 #
4185 toupper (<a>, <A>); (<b>, <B>); (<c>, <C>); (<d>, <D>); (<e>, <E>); \
4186 (<f>, <F>); (<g>, <G>); (<h>, <H>); (<i>, <I>); (<j>, <J>); \
4187 (<k>, <K>); (<l>, <L>); (<m>, <M>); (<n>, <N>); (<o>, <O>); \
4188 (<p>, <P>); (<q>, <Q>); (<r>, <R>); (<s>, <S>); (<t>, <T>); \
4189 (<u>, <U>); (<v>, <V>); (<w>, <W>); (<x>, <X>); (<y>, <Y>); (<z>, <Z>)
4190 #
4191 tolower (<A>, <a>); (<B>, <b>); (<C>, <c>); (<D>, <d>); (<E>, <e>); \
4192 (<F>, <f>); (<G>, <g>); (<H>, <h>); (<I>, <i>); (<J>, <j>); \

```

```

4193         (<K>, <k>); (<L>, <l>); (<M>, <m>); (<N>, <n>); (<O>, <o>); \
4194         (<P>, <p>); (<Q>, <q>); (<R>, <r>); (<S>, <s>); (<T>, <t>); \
4195         (<U>, <u>); (<V>, <v>); (<W>, <w>); (<X>, <x>); (<Y>, <y>); (<Z>, <z>)
4196     END LC_CTYPE
    
```

	Symbolic Name	Other Case	Character Classes
4197			
4198	<NUL>		cntrl
4199	<SOH>		cntrl
4200	<STX>		cntrl
4201	<ETX>		cntrl
4202	<EOT>		cntrl
4203	<ENQ>		cntrl
4204	<ACK>		cntrl
4205	<alert>		cntrl
4206	<backspace>		cntrl
4207	<tab>		cntrl, space, blank
4208	<newline>		cntrl, space
4209	<vertical-tab>		cntrl, space
4210	<form-feed>		cntrl, space
4211	<carriage-return>		cntrl, space
4212	<SO>		cntrl
4213	<SI>		cntrl
4214	<DLE>		cntrl
4215	<DC1>		cntrl
4216	<DC2>		cntrl
4217	<DC3>		cntrl
4218	<DC4>		cntrl
4219	<NAK>		cntrl
4220	<SYN>		cntrl
4221	<ETB>		cntrl
4222	<CAN>		cntrl
4223	<EM>		cntrl
4224	<SUB>		cntrl
4225	<ESC>		cntrl
4226	<IS4>		cntrl
4227	<IS3>		cntrl
4228	<IS2>		cntrl
4229	<IS1>		cntrl
4230	<space>		space, print, blank
4231	<exclamation-mark>		punct, print, graph
4232	<quotation-mark>		punct, print, graph
4233	<number-sign>		punct, print, graph
4234	<dollar-sign>		punct, print, graph
4235	<percent-sign>		punct, print, graph
4236	<ampersand>		punct, print, graph
4237	<apostrophe>		punct, print, graph
4238	<left-parenthesis>		punct, print, graph
4239	<right-parenthesis>		punct, print, graph
4240	<asterisk>		punct, print, graph
4241	<plus-sign>		punct, print, graph
4242	<comma>		punct, print, graph
4243	<hyphen>		punct, print, graph

## Locale

## Locale Definition

	Symbolic Name	Other Case	Character Classes
4244			
4245	<period>		punct, print, graph
4246	<slash>		punct, print, graph
4247	<zero>		digit, xdigit, print, graph
4248	<one>		digit, xdigit, print, graph
4249	<two>		digit, xdigit, print, graph
4250	<three>		digit, xdigit, print, graph
4251	<four>		digit, xdigit, print, graph
4252	<five>		digit, xdigit, print, graph
4253	<six>		digit, xdigit, print, graph
4254	<seven>		digit, xdigit, print, graph
4255	<eight>		digit, xdigit, print, graph
4256	<nine>		digit, xdigit, print, graph
4257	<colon>		punct, print, graph
4258	<semicolon>		punct, print, graph
4259	<less-than-sign>		punct, print, graph
4260	<equals-sign>		punct, print, graph
4261	<greater-than-sign>		punct, print, graph
4262	<question-mark>		punct, print, graph
4263	<commercial-at>		punct, print, graph
4264	<A>	<a>	upper, xdigit, alpha, print, graph
4265	<B>	<b>	upper, xdigit, alpha, print, graph
4266	<C>	<c>	upper, xdigit, alpha, print, graph
4267	<D>	<d>	upper, xdigit, alpha, print, graph
4268	<E>	<e>	upper, xdigit, alpha, print, graph
4269	<F>	<f>	upper, xdigit, alpha, print, graph
4270	<G>	<g>	upper, alpha, print, graph
4271	<H>	<h>	upper, alpha, print, graph
4272	<I>	<i>	upper, alpha, print, graph
4273	<J>	<j>	upper, alpha, print, graph
4274	<K>	<k>	upper, alpha, print, graph
4275	<L>	<l>	upper, alpha, print, graph
4276	<M>	<m>	upper, alpha, print, graph
4277	<N>	<n>	upper, alpha, print, graph
4278	<O>	<o>	upper, alpha, print, graph
4279	<P>	<p>	upper, alpha, print, graph
4280	<Q>	<q>	upper, alpha, print, graph
4281	<R>	<r>	upper, alpha, print, graph
4282	<S>	<s>	upper, alpha, print, graph
4283	<T>	<t>	upper, alpha, print, graph
4284	<U>	<u>	upper, alpha, print, graph
4285	<V>	<v>	upper, alpha, print, graph
4286	<W>	<w>	upper, alpha, print, graph
4287	<X>	<x>	upper, alpha, print, graph
4288	<Y>	<y>	upper, alpha, print, graph
4289	<Z>	<z>	upper, alpha, print, graph
4290	<left-square-bracket>		punct, print, graph
4291	<backslash>		punct, print, graph
4292	<right-square-bracket>		punct, print, graph
4293	<circumflex>		punct, print, graph
4294	<underscore>		punct, print, graph
4295	<grave-accent>		punct, print, graph

	Symbolic Name	Other Case	Character Classes
4296			
4297	<a>	<A>	lower, xdigit, alpha, print, graph
4298	<b>	<B>	lower, xdigit, alpha, print, graph
4299	<c>	<C>	lower, xdigit, alpha, print, graph
4300	<d>	<D>	lower, xdigit, alpha, print, graph
4301	<e>	<E>	lower, xdigit, alpha, print, graph
4302	<f>	<F>	lower, xdigit, alpha, print, graph
4303	<g>	<G>	lower, alpha, print, graph
4304	<h>	<H>	lower, alpha, print, graph
4305	<i>	<I>	lower, alpha, print, graph
4306	<j>	<J>	lower, alpha, print, graph
4307	<k>	<K>	lower, alpha, print, graph
4308	<l>	<L>	lower, alpha, print, graph
4309	<m>	<M>	lower, alpha, print, graph
4310	<n>	<N>	lower, alpha, print, graph
4311	<o>	<O>	lower, alpha, print, graph
4312	<p>	<P>	lower, alpha, print, graph
4313	<q>	<Q>	lower, alpha, print, graph
4314	<r>	<R>	lower, alpha, print, graph
4315	<s>	<S>	lower, alpha, print, graph
4316	<t>	<T>	lower, alpha, print, graph
4317	<u>	<U>	lower, alpha, print, graph
4318	<v>	<V>	lower, alpha, print, graph
4319	<w>	<W>	lower, alpha, print, graph
4320	<x>	<X>	lower, alpha, print, graph
4321	<y>	<Y>	lower, alpha, print, graph
4322	<z>	<Z>	lower, alpha, print, graph
4323	<left-curly-bracket>		punct, print, graph
4324	<vertical-line>		punct, print, graph
4325	<right-curly-bracket>		punct, print, graph
4326	<tilde>		punct, print, graph
4327	<DEL>		cntrl

4328 **7.3.2 LC\_COLLATE**

4329 The *LC\_COLLATE* category provides a collation sequence definition for numerous utilities in the  
 4330 Shell and Utilities volume of POSIX.1-2008 (*sort*, *uniq*, and so on), regular expression matching  
 4331 (see Chapter 9, on page 181), and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()* functions in the  
 4332 System Interfaces volume of POSIX.1-2008.

4333 A collation sequence definition shall define the relative order between collating elements  
 4334 (characters and multi-character collating elements) in the locale. This order is expressed in terms  
 4335 of collation values; that is, by assigning each element one or more collation values (also known  
 4336 as collation weights). This does not imply that implementations shall assign such values, but  
 4337 that ordering of strings using the resultant collation definition in the locale behaves as if such  
 4338 assignment is done and used in the collation process. At least the following capabilities are  
 4339 provided:

- 4340 1. **Multi-character collating elements.** Specification of multi-character collating elements  
 4341 (that is, sequences of two or more characters to be collated as an entity).

- 4342 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a  
 4343 collation value defining its order in the character (or basic) collation sequence. This  
 4344 ordering is used by regular expressions and pattern matching and, unless collation  
 4345 weights are explicitly specified, also as the collation weight to be used in sorting.
- 4346 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or  
 4347 more (up to the limit {COLL\_WEIGHTS\_MAX}, as defined in `<limits.h>`) collating  
 4348 weights for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4349 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4350 5. **Equivalence class definition.** Two or more collating elements have the same collation  
 4351 value (primary weight).
- 4352 6. **Ordering by weights.** When two strings are compared to determine their relative order,  
 4353 the two strings are first broken up into a series of collating elements; the elements in each  
 4354 successive pair of elements are then compared according to the relative primary weights  
 4355 for the elements. If equal, and more than one weight has been assigned, then the pairs of  
 4356 collating elements are re-compared according to the relative subsequent weights, until  
 4357 either a pair of collating elements compare unequal or the weights are exhausted.

4358 The following keywords shall be recognized in a collation sequence definition. They are  
 4359 described in detail in the following sections.

4360	<b>copy</b>	Specify the name of an existing locale which shall be used as the
4361		definition of this category. If this keyword is specified, no other keyword
4362		shall be specified.
4363	<b>collating-element</b>	Define a collating-element symbol representing a multi-character
4364		collating element. This keyword is optional.
4365	<b>collating-symbol</b>	Define a collating symbol for use in collation order statements. This
4366		keyword is optional.
4367	<b>order_start</b>	Define collation rules. This statement shall be followed by one or more
4368		collation order statements, assigning character collation values and
4369		collation weights to collating elements.
4370	<b>order_end</b>	Specify the end of the collation-order statements.

#### 4371 7.3.2.1 The collating-element Keyword

4372 In addition to the collating elements in the character set, the **collating-element** keyword can be  
 4373 used to define multi-character collating elements. The syntax is as follows:

```
4374 "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4375 The `<collating-symbol>` operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A `<collating-element>` defined via this keyword is only recognized with the `LC_COLLATE` category.

4380 For example:

```
4381 collating-element <ch> from "<c><h>"
4382 collating-element <e-acute> from "<acute><e>"
4383 collating-element <ll> from "ll"
```

4384 7.3.2.2 *The collating-symbol Keyword*

4385 This keyword shall be used to define symbols for use in collation sequence statements; that is,  
4386 between the **order\_start** and the **order\_end** keywords. The syntax is as follows:

4387 `"collating-symbol %s\n", <collating-symbol>`

4388 The `<collating-symbol>` shall be a symbolic name, enclosed between angle brackets ('<' and  
4389 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any  
4390 other symbolic name defined in this collation definition. A `<collating-symbol>` defined via this  
4391 keyword is only recognized within the `LC_COLLATE` category.

4392 For example:

4393 `collating-symbol <UPPER_CASE>`

4394 `collating-symbol <HIGH>`

4395 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative  
4396 position in the character order sequence. While such a symbolic name does not represent any  
4397 collating element, it can be used as a weight.

4398 7.3.2.3 *The order\_start Keyword*

4399 The **order\_start** keyword shall precede collation order entries and also define the number of  
4400 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4401 `"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...`

4402 The operands to the **order\_start** keyword are optional. If present, the operands define rules to be  
4403 applied when strings are compared. The number of operands define how many weights each  
4404 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the  
4405 first operand defines rules to be applied when comparing strings using the first (primary)  
4406 weight; the second when comparing strings using the second weight, and so on. Operands shall  
4407 be separated by `<semicolon>` characters (';'). Each operand shall consist of one or more  
4408 collation directives, separated by `<comma>` characters (',' ). If the number of operands exceeds  
4409 the `{COLL_WEIGHTS_MAX}` limit, the utility shall issue a warning message. The following  
4410 directives shall be supported:

4411 **forward** Specifies that comparison operations for the weight level shall proceed from start  
4412 of string towards the end of string.

4413 **backward** Specifies that comparison operations for the weight level shall proceed from end of  
4414 string towards the beginning of string.

4415 **position** Specifies that comparison operations for the weight level shall consider the relative  
4416 position of elements in the strings not subject to **IGNORE**. The string containing  
4417 an element not subject to **IGNORE** after the fewest collating elements subject to  
4418 **IGNORE** from the start of the compare shall collate first. If both strings contain a  
4419 character not subject to **IGNORE** in the same relative position, the collating values  
4420 assigned to the elements shall determine the ordering. In case of equality,  
4421 subsequent characters not subject to **IGNORE** shall be considered in the same  
4422 manner.

4423 The directives **forward** and **backward** are mutually-exclusive.

4424 If no operands are specified, a single **forward** operand shall be assumed.

4425 For example:

4426 `order_start forward;backward`

#### 4427 7.3.2.4 Collation Order

4428 The **order\_start** keyword shall be followed by collating identifier entries. The syntax for the  
4429 collating element entries is as follows:

4430 `"%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...`

4431 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section  
4432 7.3, on page 136), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol  
4433 UNDEFINED. The order in which collating elements are specified determines the character  
4434 order sequence, such that each collating element shall compare less than the elements following  
4435 it.

4436 A *collating-element* shall be used to specify multi-character collating elements, and indicates  
4437 that the character sequence specified via the *collating-element* is to be collated as a unit and in  
4438 the relative order specified by its place.

4439 A *collating-symbol* can be used to define a position in the relative order for use in weights. No  
4440 weights shall be specified with a *collating-symbol*.

4441 The ellipsis symbol specifies that a sequence of characters shall collate according to their  
4442 encoded character values. It shall be interpreted as indicating that all characters with a coded  
4443 character set value higher than the value of the character in the preceding line, and lower than  
4444 the coded character set value for the character in the following line, in the current coded  
4445 character set, shall be placed in the character collation order between the previous and the  
4446 following character in ascending order according to their coded character set values. An initial  
4447 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing  
4448 ellipsis as if the following line specified the highest coded character set value in the current  
4449 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do  
4450 not specify characters in the current coded character set. The use of the ellipsis symbol ties the  
4451 definition to a specific coded character set and may preclude the definition from being portable  
4452 between implementations.

4453 The symbol UNDEFINED shall be interpreted as including all coded character set values not  
4454 specified explicitly, or via the ellipsis symbol. Such characters shall be inserted in the character  
4455 collation order at the point indicated by the symbol, and in ascending order according to their  
4456 coded character set values. If no UNDEFINED symbol is specified, and the current coded  
4457 character set contains characters not specified in this section, the utility shall issue a warning  
4458 message and place such characters at the end of the character collation order.

4459 The optional operands for each collation-element shall be used to define the primary, secondary,  
4460 or subsequent weights for the collating element. The first operand specifies the relative primary  
4461 weight, the second the relative secondary weight, and so on. Two or more collation-elements can  
4462 be assigned the same weight; they belong to the same "equivalence class" if they have the same  
4463 primary weight. Collation shall behave as if, for each weight level, elements subject to IGNORE  
4464 are removed, unless the **position** collation directive is specified for the corresponding level with  
4465 the **order\_start** keyword. Then each successive pair of elements shall be compared according to  
4466 the relative weights for the elements. If the two strings compare equal, the process shall be  
4467 repeated for the next weight level, up to the limit {COLL\_WEIGHTS\_MAX}.

4468 Weights shall be expressed as characters (in any of the forms specified in Section 7.3, on page  
4469 136), *collating-symbol*s, *collating-element*s, an ellipsis, or the special symbol IGNORE. A  
4470 single character, a *collating-symbol*, or a *collating-element* shall represent the relative position

4471 in the character collating sequence of the character or symbol, rather than the character or  
 4472 characters themselves. Thus, rather than assigning absolute values to weights, a particular  
 4473 weight is expressed using the relative order value assigned to a collating element based on its  
 4474 order in the character collation sequence.

4475 One-to-many mapping is indicated by specifying two or more concatenated characters or  
 4476 symbolic names. For example, if the <eszt> is given the string "<s><s>" as a weight,  
 4477 comparisons are performed as if all occurrences of the <eszt> are replaced by "<s><s>"  
 4478 (assuming that "<s>" has the collating weight "<s>"). If it is necessary to define <eszt> and  
 4479 "<s><s>" as an equivalence class, then a collating element must be defined for the string "ss".

4480 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the  
 4481 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special  
 4482 symbol shall by default be assigned the same primary weight (that is, they belong to the same  
 4483 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each  
 4484 character in the sequence shall have unique weights, equal to the relative order of their character  
 4485 in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error  
 4486 if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4487 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using  
 4488 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that  
 4489 is, as if the string did not contain the collating element. In regular expressions and pattern  
 4490 matching, all characters that are subject to **IGNORE** in their primary weight form an  
 4491 equivalence class.

4492 An empty operand shall be interpreted as the collating element itself.

4493 For example, the order statement:

4494 <a> <a>;<a>

4495 is equal to:

4496 <a>

4497 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be  
 4498 interpreted as the value of each character defined by the ellipsis.

4499 The collation order as defined in this section affects the interpretation of bracket expressions in  
 4500 regular expressions (see Section 9.3.5, on page 184).

4501 For example:

4502	order_start	forward;backward
4503	UNDEFINED	IGNORE; IGNORE
4504	<LOW>	
4505	<space>	<LOW>;<space>
4506		<LOW>;...
4507	<a>	<a>;<a>
4508	<a-acute>	<a>;<a-acute>
4509	<a-grave>	<a>;<a-grave>
4510	<A>	<a>;<A>
4511	<A-acute>	<a>;<A-acute>
4512	<A-grave>	<a>;<A-grave>
4513	<ch>	<ch>;<ch>
4514	<Ch>	<ch>;<Ch>
4515	<s>	<s>;<s>
4516	<eszt>	"<s><s>"; "<eszt><eszt>"
4517	order_end	

4518 This example is interpreted as follows:

- 4519 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or  
4520 via the ellipsis) shall be ignored for collation purposes.
- 4521 2. All characters between <space> and ' a ' shall have the same primary equivalence class  
4522 and individual secondary weights based on their ordinal encoded values.
- 4523 3. All characters based on the uppercase or lowercase character ' a ' belong to the same  
4524 primary equivalence class.
- 4525 4. The multi-character collating element <ch> is represented by the collating symbol <ch>  
4526 and belongs to the same primary equivalence class as the multi-character collating  
4527 element <Ch>.

4528 7.3.2.5 *The order\_end Keyword*

4529 The collating order entries shall be terminated with an **order\_end** keyword.

4530 7.3.2.6 *LC\_COLLATE Category in the POSIX Locale*

4531 The collation sequence definition of the POSIX locale follows; the code listing depicts the  
4532 *localedef* input.

```
4533 LC_COLLATE
4534 # This is the POSIX locale definition for the LC_COLLATE category.
4535 # The order is the same as in the ASCII codeset.
4536 order_start forward
4537 <NUL>
4538 <SOH>
4539 <STX>
4540 <ETX>
4541 <EOT>
4542 <ENQ>
4543 <ACK>
4544 <alert>
4545 <backspace>
4546 <tab>
4547 <newline>
4548 <vertical-tab>
4549 <form-feed>
4550 <carriage-return>
4551 <SO>
4552 <SI>
4553 <DLE>
4554 <DC1>
4555 <DC2>
4556 <DC3>
4557 <DC4>
4558 <NAK>
4559 <SYN>
4560 <ETB>
4561 <CAN>
4562 <EM>
4563 <SUB>
```

*Locale Definition**Locale*

4564	<ESC>
4565	<IS4>
4566	<IS3>
4567	<IS2>
4568	<IS1>
4569	<space>
4570	<exclamation-mark>
4571	<quotation-mark>
4572	<number-sign>
4573	<dollar-sign>
4574	<percent-sign>
4575	<ampersand>
4576	<apostrophe>
4577	<left-parenthesis>
4578	<right-parenthesis>
4579	<asterisk>
4580	<plus-sign>
4581	<comma>
4582	<hyphen>
4583	<period>
4584	<slash>
4585	<zero>
4586	<one>
4587	<two>
4588	<three>
4589	<four>
4590	<five>
4591	<six>
4592	<seven>
4593	<eight>
4594	<nine>
4595	<colon>
4596	<semicolon>
4597	<less-than-sign>
4598	<equals-sign>
4599	<greater-than-sign>
4600	<question-mark>
4601	<commercial-at>
4602	<A>
4603	<B>
4604	<C>
4605	<D>
4606	<E>
4607	<F>
4608	<G>
4609	<H>
4610	<I>
4611	<J>
4612	<K>
4613	<L>
4614	<M>
4615	<N>
4616	<O>

*Locale**Locale Definition*

4617 <P>  
 4618 <Q>  
 4619 <R>  
 4620 <S>  
 4621 <T>  
 4622 <U>  
 4623 <V>  
 4624 <W>  
 4625 <X>  
 4626 <Y>  
 4627 <Z>  
 4628 <left-square-bracket>  
 4629 <backslash>  
 4630 <right-square-bracket>  
 4631 <circumflex>  
 4632 <underscore>  
 4633 <grave-accent>  
 4634 <a>  
 4635 <b>  
 4636 <c>  
 4637 <d>  
 4638 <e>  
 4639 <f>  
 4640 <g>  
 4641 <h>  
 4642 <i>  
 4643 <j>  
 4644 <k>  
 4645 <l>  
 4646 <m>  
 4647 <n>  
 4648 <o>  
 4649 <p>  
 4650 <q>  
 4651 <r>  
 4652 <s>  
 4653 <t>  
 4654 <u>  
 4655 <v>  
 4656 <w>  
 4657 <x>  
 4658 <y>  
 4659 <z>  
 4660 <left-curly-bracket>  
 4661 <vertical-line>  
 4662 <right-curly-bracket>  
 4663 <tilde>  
 4664 <DEL>  
 4665 order\_end  
 4666 #  
 4667 END LC\_COLLATE

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

4668 **7.3.3 LC\_MONETARY**

4669 The *LC\_MONETARY* category shall define the rules and symbols that are used to format  
4670 monetary numeric information.

4671 This information is available through the *localeconv()* function and is used by the *strfmon()*  
4672 function.

4673 Some of the information is also available in an alternative form via the *nl\_langinfo()* function  
4674 (see CRNCYSTR in [<langinfo.h>](#)).

4675 The following items are defined in this category of the locale. The item names are the keywords  
4676 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
4677 names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the  
4678 header. The *localeconv()* function returns {CHAR\_MAX} for unspecified integer items and the  
4679 empty string (" ") for unspecified or size zero string items.

4680 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
4681 [Section 7.4](#) (on page 165). For some keywords, the strings can contain only integers. Keywords  
4682 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,  
4683 are used to indicate that the value is not available in the locale. The following keywords shall be  
4684 recognized:

4685 **copy** Specify the name of an existing locale which shall be used as the  
4686 definition of this category. If this keyword is specified, no other keyword  
4687 shall be specified.

4688 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4689 **int\_curr\_symbol** The international currency symbol. The operand shall be a four-character  
4690 string, with the first three characters containing the alphabetic  
4691 international currency symbol. The international currency symbol should  
4692 be chosen in accordance with those specified in the ISO 4217 standard.  
4693 The fourth character shall be the character used to separate the  
4694 international currency symbol from the monetary quantity.

4695 **currency\_symbol** The string that shall be used as the local currency symbol.

4696 **mon\_decimal\_point** The operand is a string containing the symbol that shall be used as the  
4697 decimal delimiter (radix character) in monetary formatted quantities.

4698 **mon\_thousands\_sep** The operand is a string containing the symbol that shall be used as a  
4699 separator for groups of digits to the left of the decimal delimiter in  
4700 formatted monetary quantities.

4701 **mon\_grouping** Define the size of each group of digits in formatted monetary quantities.  
4702 The operand is a sequence of integers separated by [<semicolon>](#)  
4703 characters. Each integer specifies the number of digits in each group, with  
4704 the initial integer defining the size of the group immediately preceding  
4705 the decimal delimiter, and the following integers defining the preceding  
4706 groups. If the last integer is not -1, then the size of the previous group (if  
4707 any) shall be repeatedly used for the remainder of the digits. If the last  
4708 integer is -1, then no further grouping shall be performed.

4709 **positive\_sign** A string that shall be used to indicate a non-negative-valued formatted  
4710 monetary quantity.

4711 **negative\_sign** A string that shall be used to indicate a negative-valued formatted  
4712 monetary quantity.

## Locale

## Locale Definition

4713	<b>int_frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>int_curr_symbol</b> .
4714		
4715		
4716	<b>frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>currency_symbol</b> .
4717		
4718		
4719	<b>p_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4720		
4721		
4722	<b>p_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a non-negative formatted monetary quantity.
4723		
4724		The values of <b>p_sep_by_space</b> , <b>n_sep_by_space</b> , <b>int_p_sep_by_space</b> , and <b>int_n_sep_by_space</b> are interpreted according to the following:
4725		
4726		0 No <space> separates the currency symbol and value.
4727		1 If the currency symbol and sign string are adjacent, a <space> separates them from the value; otherwise, a <space> separates the currency symbol from the value.
4728		
4729		
4730		2 If the currency symbol and sign string are adjacent, a <space> separates them; otherwise, a <space> separates the sign string from the value.
4731		
4732		
4733	<b>n_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4734		
4735		
4736	<b>n_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a negative formatted monetary quantity.
4737		
4738	<b>p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a monetary quantity with a non-negative value. The following integer values shall be recognized for <b>int_n_sign_posn</b> , <b>int_p_sign_posn</b> , <b>n_sign_posn</b> , and <b>p_sign_posn</b> :
4739		
4740		
4741		
4742		0 Parentheses enclose the quantity and the <b>currency_symbol</b> .
4743		1 The sign string precedes the quantity and the <b>currency_symbol</b> .
4744		2 The sign string succeeds the quantity and the <b>currency_symbol</b> .
4745		3 The sign string precedes the <b>currency_symbol</b> .
4746		4 The sign string succeeds the <b>currency_symbol</b> .
4747	<b>n_sign_posn</b>	An integer set to a value indicating the positioning of the <b>negative_sign</b> for a negative formatted monetary quantity.
4748		
4749	<b>int_p_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4750		
4751		
4752	<b>int_n_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4753		
4754		

4755	<b>int_p_sep_by_space</b>	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the value for a non-negative internationally formatted monetary quantity.
4756		
4757		
4758	<b>int_n_sep_by_space</b>	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the value for a negative internationally formatted monetary quantity.
4759		
4760		
4761	<b>int_p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a positive monetary quantity formatted with the international format.
4762		
4763	<b>int_n_sign_posn</b>	An integer set to a value indicating the positioning of the <b>negative_sign</b> for a negative monetary quantity formatted with the international format.
4764		

### 4765 7.3.3.1 LC\_MONETARY Category in the POSIX Locale

4766 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the  
 4767 *localedef* input, the table representing the same information with the addition of *localeconv()* and  
 4768 *nl\_langinfo()* formats. All values are unspecified in the POSIX locale.

```

4769 LC_MONETARY
4770 # This is the POSIX locale definition for
4771 # the LC_MONETARY category.
4772 #
4773 int_curr_symbol      ""
4774 currency_symbol     ""
4775 mon_decimal_point   ""
4776 mon_thousands_sep  ""
4777 mon_grouping        -1
4778 positive_sign       ""
4779 negative_sign       ""
4780 int_frac_digits     -1
4781 frac_digits         -1
4782 p_cs_precedes       -1
4783 p_sep_by_space      -1
4784 n_cs_precedes       -1
4785 n_sep_by_space      -1
4786 p_sign_posn        -1
4787 n_sign_posn         -1
4788 int_p_cs_precedes   -1
4789 int_p_sep_by_space  -1
4790 int_n_cs_precedes   -1
4791 int_n_sep_by_space  -1
4792 int_p_sign_posn     -1
4793 int_n_sign_posn     -1
4794 #
4795 END LC_MONETARY

```

	Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
4796	<b>int_curr_symbol</b>	—	N/A	" "	" "
4797	<b>currency_symbol</b>	CRNCYSTR	N/A	" "	" "
4798	<b>mon_decimal_point</b>	—	N/A	" "	" "
4800	<b>mon_thousands_sep</b>	—	N/A	" "	" "
4801	<b>mon_grouping</b>	—	N/A	" "	-1
4802	<b>positive_sign</b>	—	N/A	" "	" "
4803	<b>negative_sign</b>	—	N/A	" "	" "
4804	<b>int_frac_digits</b>	—	N/A	{CHAR_MAX}	-1
4805	<b>frac_digits</b>	—	N/A	{CHAR_MAX}	-1
4806	<b>p_cs_precedes</b>	CRNCYSTR	N/A	{CHAR_MAX}	-1
4807	<b>p_sep_by_space</b>	—	N/A	{CHAR_MAX}	-1
4808	<b>n_cs_precedes</b>	CRNCYSTR	N/A	{CHAR_MAX}	-1
4809	<b>n_sep_by_space</b>	—	N/A	{CHAR_MAX}	-1
4810	<b>p_sign_posn</b>	—	N/A	{CHAR_MAX}	-1
4811	<b>n_sign_posn</b>	—	N/A	{CHAR_MAX}	-1
4812	<b>int_p_cs_precedes</b>	—	N/A	{CHAR_MAX}	-1
4813	<b>int_p_sep_by_space</b>	—	N/A	{CHAR_MAX}	-1
4814	<b>int_n_cs_precedes</b>	—	N/A	{CHAR_MAX}	-1
4815	<b>int_n_sep_by_space</b>	—	N/A	{CHAR_MAX}	-1
4816	<b>int_p_sign_posn</b>	—	N/A	{CHAR_MAX}	-1
4817	<b>int_n_sign_posn</b>	—	N/A	{CHAR_MAX}	-1

4819 The entry N/A indicates that the value is not available in the POSIX locale.

### 4820 7.3.4 LC\_NUMERIC

4821 The *LC\_NUMERIC* category shall define the rules and symbols that are used to format non-  
4822 monetary numeric information. This information is available through the *localeconv()* function.

4823 Some of the information is also available in an alternative form via the *nl\_langinfo()* function.

4824 The following items are defined in this category of the locale. The item names are the keywords  
4825 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
4826 names of the *lconv* structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the  
4827 header. The *localeconv()* function returns {CHAR\_MAX} for unspecified integer items and the  
4828 empty string (" ") for unspecified or size zero string items.

4829 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
4830 Section 7.4 (on page 165). For some keywords, the strings can only contain integers. Keywords  
4831 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,  
4832 shall be used to indicate that the value is not available in the locale. The following keywords  
4833 shall be recognized:

4834 **copy** Specify the name of an existing locale which shall be used as the definition of  
4835 this category. If this keyword is specified, no other keyword shall be specified.

4836 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4837 **decimal\_point** The operand is a string containing the symbol that shall be used as the  
4838 decimal delimiter (radix character) in numeric, non-monetary formatted  
4839 quantities. This keyword cannot be omitted and cannot be set to the empty  
4840 string. In contexts where standards limit the **decimal\_point** to a single byte,  
4841 the result of specifying a multi-byte operand shall be unspecified.

4842        **thousands\_sep**    The operand is a string containing the symbol that shall be used as a separator  
 4843        for groups of digits to the left of the decimal delimiter in numeric, non-  
 4844        monetary formatted monetary quantities. In contexts where standards limit  
 4845        the **thousands\_sep** to a single byte, the result of specifying a multi-byte  
 4846        operand shall be unspecified.

4847        **grouping**            Define the size of each group of digits in formatted non-monetary quantities.  
 4848        The operand is a sequence of integers separated by <semicolon> characters.  
 4849        Each integer specifies the number of digits in each group, with the initial  
 4850        integer defining the size of the group immediately preceding the decimal  
 4851        delimiter, and the following integers defining the preceding groups. If the last  
 4852        integer is not -1, then the size of the previous group (if any) shall be  
 4853        repeatedly used for the remainder of the digits. If the last integer is -1, then no  
 4854        further grouping shall be performed.

4855    7.3.4.1    *LC\_NUMERIC Category in the POSIX Locale*

4856        The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing  
 4857        depicting the *localedef* input, the table representing the same information with the addition of  
 4858        *localeconv()* values, and *nl\_langinfo()* constants.

```

4859    LC_NUMERIC
4860    # This is the POSIX locale definition for
4861    # the LC_NUMERIC category.
4862    #
4863    decimal_point        "<period>"
4864    thousands_sep        ""
4865    grouping             -1
4866    #
4867    END LC_NUMERIC
    
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
<b>decimal_point</b>	RADIXCHAR	."	."	.
<b>thousands_sep</b>	THOUSEP	N/A	""	""
<b>grouping</b>	—	N/A	""	-1

4873        The entry N/A indicates that the value is not available in the POSIX locale.

4874    7.3.5    **LC\_TIME**

4875        The *LC\_TIME* category shall define the interpretation of the conversion specifications supported  
 4876        by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and  
 4877        *nl\_langinfo()* functions. Since the interfaces for C-language access and locale definition differ  
 4878        significantly, they are described separately.

4879    7.3.5.1    *LC\_TIME Locale Definition*

4880        In a locale definition, the following mandatory keywords shall be recognized:

4881        **copy**                Specify the name of an existing locale which shall be used as the definition of  
 4882        this category. If this keyword is specified, no other keyword shall be specified.

## Locale

## Locale Definition

4883	<b>abday</b>	Define the abbreviated weekday names, corresponding to the %a conversion specification (conversion specification in the <i>strftime()</i> , <i>wcsftime()</i> , and <i>strptime()</i> functions). The operand shall consist of seven <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the day corresponding to Sunday, the second the abbreviated name of the day corresponding to Monday, and so on.
4884		
4885		
4886		
4887		
4888		
4889	<b>day</b>	Define the full weekday names, corresponding to the %A conversion specification. The operand shall consist of seven <semicolon>-separated strings, each surrounded by double-quotes. The first string is the full name of the day corresponding to Sunday, the second the full name of the day corresponding to Monday, and so on.
4890		
4891		
4892		
4893		
4894	<b>abmon</b>	Define the abbreviated month names, corresponding to the %b conversion specification. The operand shall consist of twelve <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.
4895		
4896		
4897		
4898		
4899	<b>mon</b>	Define the full month names, corresponding to the %B conversion specification. The operand shall consist of twelve <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the full name of the first month of the year (January), the second the full name of the second month, and so on.
4900		
4901		
4902		
4903		
4904	<b>d_t_fmt</b>	Define the appropriate date and time representation, corresponding to the %c conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
4905		
4906		
4907		
4908		
4909	<b>d_fmt</b>	Define the appropriate date representation, corresponding to the %x conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
4910		
4911		
4912		
4913	<b>t_fmt</b>	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
4914		
4915		
4916		
4917	<b>am_pm</b>	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i> strings, corresponding to the %p conversion specification. The operand shall consist of two strings, separated by a <semicolon>, each surrounded by double-quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last string the <i>post-meridiem</i> designation.
4918		
4919		
4920		
4921		
4922	<b>t_fmt_ampm</b>	Define the appropriate time representation in the 12-hour clock format with <b>am_pm</b> , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
4923		
4924		
4925		
4926		
4927	<b>era</b>	Define how years are counted and displayed for each era in a locale. The operand shall consist of <semicolon>-separated strings. Each string shall be an era description segment with the format:
4928		
4929		
4930		<i>direction:offset:start_date:end_date:era_name:era_format</i>

4931		according to the definitions below. There can be as many era description
4932		segments as are necessary to describe the different eras.
4933	<b>Note:</b>	The start of an era might not be the earliest point in the era—it may be the
4934		latest. For example, the Christian era BC starts on the day before January 1,
4935		AD 1, and increases with earlier time.
4936	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
4937		that years closer to the <i>start_date</i> have lower numbers than those
4938		closer to the <i>end_date</i> . The '-' character shall indicate that years
4939		closer to the <i>start_date</i> have higher numbers than those closer to
4940		the <i>end_date</i> .
4941	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era,
4942		corresponding to the %EY conversion specification.
4943	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the
4944		year, month, and day numbers respectively of the start of the era.
4945		Years prior to AD 1 shall be represented as negative numbers.
4946	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
4947		or one of the two special values "-*" or "+*". The value "-*"
4948		shall indicate that the ending date is the beginning of time. The
4949		value "+*" shall indicate that the ending date is the end of time.
4950	<i>era_name</i>	A string representing the name of the era, corresponding to the
4951		%EC conversion specification.
4952	<i>era_format</i>	A string for formatting the year in the era, corresponding to the
4953		%EY conversion specification.
4954	<b>era_d_fmt</b>	Define the format of the date in alternative era notation, corresponding to the
4955		%Ex conversion specification.
4956	<b>era_t_fmt</b>	Define the locale's appropriate alternative time format, corresponding to the
4957		%EX conversion specification.
4958	<b>era_d_t_fmt</b>	Define the locale's appropriate alternative date and time format,
4959		corresponding to the %Ec conversion specification.
4960	<b>alt_digits</b>	Define alternative symbols for digits, corresponding to the %O modified
4961		conversion specification. The operand shall consist of <semicolon>-separated
4962		strings, each surrounded by double-quotes. The first string shall be the
4963		alternative symbol corresponding with zero, the second string the symbol
4964		corresponding with one, and so on. Up to 100 alternative symbol strings can
4965		be specified. The %O modifier shall indicate that the string corresponding to
4966		the value specified via the conversion specification shall be used instead of the
4967		value.
4968	7.3.5.2	<i>LC_TIME</i> C-Language Access
4969		The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the
4970		<i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are
4971		defined in the < <i>langinfo.h</i> > header.
4972	ABDAY_x	The abbreviated weekday names (for example, Sun), where <i>x</i> is a number
4973		from 1 to 7.

## Locale

## Locale Definition

4974	DAY_x	The full weekday names (for example, Sunday), where <i>x</i> is a number from 1 to 7.
4975		
4976	ABMON_x	The abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
4977		
4978	MON_x	The full month names (for example, January), where <i>x</i> is a number from 1 to 12.
4979		
4980	D_T_FMT	The appropriate date and time representation.
4981	D_FMT	The appropriate date representation.
4982	T_FMT	The appropriate time representation.
4983	AM_STR	The appropriate ante-meridiem affix.
4984	PM_STR	The appropriate post-meridiem affix.
4985	T_FMT_AMPM	The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.
4986		
4987	ERA	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:
4988		
4989		
4990		<i>direction:offset:start_date:end_date:era_name:era_format</i>
4991		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras. Era description segments are separated by <semicolon> characters.
4992		
4993		
4994		<i>direction</i> Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
4995		
4996		
4997		
4998		
4999		<i>offset</i> The number of the year closest to the <i>start_date</i> in the era.
5000		<i>start_date</i> A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5001		
5002		
5003		<i>end_date</i> The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5004		
5005		
5006		
5007		<i>era_name</i> The era, corresponding to the %EC conversion specification.
5008		<i>era_format</i> The format of the year in the era, corresponding to the %EY conversion specification.
5009		
5010	ERA_D_FMT	The era date format.
5011	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5012		
5013	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to the %EC conversion specification.
5014		

5015 ALT\_DIGITS The alternative symbols for digits, corresponding to the %0 conversion  
 5016 specification modifier. The value consists of <semicolon>-separated symbols.  
 5017 The first is the alternative symbol corresponding to zero, the second is the  
 5018 symbol corresponding to one, and so on. Up to 100 alternative symbols may  
 5019 be specified.

5020 7.3.5.3 LC\_TIME Category in the POSIX Locale

5021 The LC\_TIME category definition of the POSIX locale follows; the code listing depicts the  
 5022 *localedef* input; the table represents the same information with the addition of *localedef* keywords,  
 5023 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*  
 5024 functions, and *nl\_langinfo()* constants.

```

5025 LC_TIME
5026 # This is the POSIX locale definition for
5027 # the LC_TIME category.
5028 #
5029 # Abbreviated weekday names (%a)
5030 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>"; \
5031           "<T><h><u>"; "<F><r><i>"; "<S><a><t>"
5032 #
5033 # Full weekday names (%A)
5034 day        "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>"; \
5035           "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>"; \
5036           "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>"; \
5037           "<S><a><t><u><r><d><a><y>"
5038 #
5039 # Abbreviated month names (%b)
5040 abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>"; \
5041           "<A><p><r>"; "<M><a><y>"; "<J><u><n>"; \
5042           "<J><u><l>"; "<A><u><g>"; "<S><e><p>"; \
5043           "<O><c><t>"; "<N><o><v>"; "<D><e><c>"
5044 #
5045 # Full month names (%B)
5046 mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>"; \
5047           "<M><a><r><c><h>"; "<A><p><r><i><l>"; \
5048           "<M><a><y>"; "<J><u><n><e>"; \
5049           "<J><u><l><y>"; "<A><u><g><u><s><t>"; \
5050           "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>"; \
5051           "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"
5052 #
5053 # Equivalent of AM/PM (%p)      "AM"; "PM"
5054 am_pm      "<A><M>"; "<P><M>"
5055 #
5056 # Appropriate date and time representation (%c)
5057 #      "%a %b %e %H:%M:%S %Y"
5058 d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
5059 <space><percent-sign><e><space><percent-sign><H>\
5060 <colon><percent-sign><M><colon><percent-sign><S>\
5061 <space><percent-sign><Y>"
5062 #
5063 # Appropriate date representation (%x)      "%m/%d/%y"
5064 d_fmt      "<percent-sign><m><slash><percent-sign><d>\

```

## Locale

## Locale Definition

```

5065 <slash><percent-sign><y>"
5066 #
5067 # Appropriate time representation (%X) "%H:%M:%S"
5068 t_fmt "%<percent-sign><H><colon><percent-sign><M>\
5069 <colon><percent-sign><S>"
5070 #
5071 # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5072 t_fmt_ampm "%<percent-sign><I><colon><percent-sign><M><colon>\
5073 <percent-sign><S><space><percent-sign><p>"
5074 #
5075 END LC_TIME

```

5076 5077	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5078	d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
5079	d_fmt	D_FMT	%x	"%m/%d/%y"
5080	t_fmt	T_FMT	%X	"%H:%M:%S"
5081	am_pm	AM_STR	%p	"AM"
5082	am_pm	PM_STR	%p	"PM"
5083	t_fmt_ampm	T_FMT_AMPM	%r	"%I:%M:%S %p"
5084	day	DAY_1	%A	"Sunday"
5085	day	DAY_2	%A	"Monday"
5086	day	DAY_3	%A	"Tuesday"
5087	day	DAY_4	%A	"Wednesday"
5088	day	DAY_5	%A	"Thursday"
5089	day	DAY_6	%A	"Friday"
5090	day	DAY_7	%A	"Saturday"
5091	abday	ABDAY_1	%a	"Sun"
5092	abday	ABDAY_2	%a	"Mon"
5093	abday	ABDAY_3	%a	"Tue"
5094	abday	ABDAY_4	%a	"Wed"
5095	abday	ABDAY_5	%a	"Thu"
5096	abday	ABDAY_6	%a	"Fri"
5097	abday	ABDAY_7	%a	"Sat"
5098	mon	MON_1	%B	"January"
5099	mon	MON_2	%B	"February"
5100	mon	MON_3	%B	"March"
5101	mon	MON_4	%B	"April"
5102	mon	MON_5	%B	"May"
5103	mon	MON_6	%B	"June"
5104	mon	MON_7	%B	"July"
5105	mon	MON_8	%B	"August"
5106	mon	MON_9	%B	"September"
5107	mon	MON_10	%B	"October"
5108	mon	MON_11	%B	"November"
5109	mon	MON_12	%B	"December"
5110	abmon	ABMON_1	%b	"Jan"
5111	abmon	ABMON_2	%b	"Feb"
5112	abmon	ABMON_3	%b	"Mar"
5113	abmon	ABMON_4	%b	"Apr"
5114	abmon	ABMON_5	%b	"May"
5115	abmon	ABMON_6	%b	"Jun"

	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5116	<b>abmon</b>	ABMON_7	%b	"Jul "
5117	<b>abmon</b>	ABMON_8	%b	"Aug "
5118	<b>abmon</b>	ABMON_9	%b	"Sep "
5119	<b>abmon</b>	ABMON_10	%b	"Oct "
5120	<b>abmon</b>	ABMON_11	%b	"Nov "
5121	<b>abmon</b>	ABMON_12	%b	"Dec "
5122	<b>era</b>	ERA	%EC, %Ey, %EY	N/A
5123	<b>era_d_fmt</b>	ERA_D_FMT	%Ex	N/A
5124	<b>era_t_fmt</b>	ERA_T_FMT	%EX	N/A
5125	<b>era_d_t_fmt</b>	ERA_D_T_FMT	%Ec	N/A
5126	<b>alt_digits</b>	ALT_DIGITS	%O	N/A

5127 The entry N/A indicates the value is not available in the POSIX locale.

5130 **7.3.6 LC\_MESSAGES**

5131 The *LC\_MESSAGES* category shall define the format and values used by various utilities for  
 5132 affirmative and negative responses. This information is available through the *nl\_langinfo()*  
 5133 function.

5134 The message catalog used by the standard utilities and selected by the *catopen()* function shall be  
 5135 determined by the setting of *NLSPATH*; see Chapter 8 (on page 173). The *LC\_MESSAGES*  
 5136 category can be specified as part of an *NLSPATH* substitution field.

5137 The following keywords shall be recognized as part of the locale definition file.

5138 **copy** Specify the name of an existing locale which shall be used as the definition of this  
 5139 category. If this keyword is specified, no other keyword shall be specified.

5140 **Note:** This is a *localedef* keyword, unavailable through *nl\_langinfo()*.

5141 **yesexpr** The operand consists of an extended regular expression (see Section 9.4, on page  
 5142 188) that describes the acceptable affirmative response to a question expecting an  
 5143 affirmative or negative response.

5144 **noexpr** The operand consists of an extended regular expression that describes the  
 5145 acceptable negative response to a question expecting an affirmative or negative  
 5146 response.

5147 **7.3.6.1 LC\_MESSAGES Category in the POSIX Locale**

5148 The format and values for affirmative and negative responses of the POSIX locale follow; the  
 5149 code listing depicting the *localedef* input, the table representing the same information with the  
 5150 addition of *nl\_langinfo()* constants.

```

5151 LC_MESSAGES
5152 # This is the POSIX locale definition for
5153 # the LC_MESSAGES category.
5154 #
5155 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5156 #
5157 noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5158 #
5159 END LC_MESSAGES
    
```

	localedef Keyword	langinfo Constant	POSIX Locale Value
5160			
5161	<b>yesexpr</b>	YESEXPR	"^[yY]"
5162	<b>noexpr</b>	NOEXPR	"^[nN]"

## 5163 7.4 Locale Definition Grammar

5164 The grammar and lexical conventions in this section shall together describe the syntax for the  
 5165 locale definition source. The general conventions for this style of grammar are described in XCU  
 5166 [Section 1.3](#) (on page 2287). The grammar shall take precedence over the text in this chapter.

### 5167 7.4.1 Locale Lexical Conventions

5168 The lexical conventions for the locale definition grammar are described in this section.

5169 The following tokens shall be processed (in addition to those string constants shown in the  
 5170 grammar):

5171	<b>LOC_NAME</b>	A string of characters representing the name of a locale.
5172	<b>CHAR</b>	Any single character.
5173	<b>NUMBER</b>	A decimal number, represented by one or more decimal digits.
5174	<b>COLLSYMBOL</b>	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a <b>COLLELEMENT</b> symbol.
5177	<b>COLLELEMENT</b>	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a <b>COLLSYMBOL</b> symbol.
5179	<b>CHARCLASS</b>	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5183	<b>CHARSYMBOL</b>	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5185	<b>OCTAL_CHAR</b>	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a <backslash>) followed by two or more octal digits.
5189	<b>HEX_CHAR</b>	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant <i>x</i> and two or more hexadecimal digits.
5193	<b>DECIMAL_CHAR</b>	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.

5197	<b>ELLIPSIS</b>	The string "...".
5198	<b>EXTENDED_REG_EXP</b>	An extended regular expression as defined in the grammar in <a href="#">Section 9.5</a> (on page 191).
5199		
5200	<b>EOL</b>	The line termination character <newline>.

## 5201 7.4.2 Locale Grammar

5202 This section presents the grammar for the locale definition.

5203	%token	LOC_NAME
5204	%token	CHAR
5205	%token	NUMBER
5206	%token	COLLSYMBOL COLLELEMENT
5207	%token	CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5208	%token	ELLIPSIS
5209	%token	EXTENDED_REG_EXP
5210	%token	EOL
5211	%start	locale_definition
5212	%%	
5213	locale_definition	: global_statements locale_categories
5214		locale_categories
5215		;
5216	global_statements	: global_statements symbol_redefine
5217		symbol_redefine
5218		;
5219	symbol_redefine	: 'escape_char' CHAR EOL
5220		'comment_char' CHAR EOL
5221		;
5222	locale_categories	: locale_categories locale_category
5223		locale_category
5224		;
5225	locale_category	: lc_ctype   lc_collate   lc_messages
5226		lc_monetary   lc_numeric   lc_time
5227		;
5228		/* The following grammar rules are common to all categories */
5229	char_list	: char_list char_symbol
5230		char_symbol
5231		;
5232	char_symbol	: CHAR   CHARSYMBOL
5233		OCTAL_CHAR   HEX_CHAR   DECIMAL_CHAR
5234		;
5235	elem_list	: elem_list char_symbol
5236		elem_list COLLSYMBOL
5237		elem_list COLLELEMENT
5238		char_symbol
5239		COLLSYMBOL

## Locale

## Locale Definition Grammar

```

5240         | COLLELEMENT
5241         ;
5242     symb_list      : symb_list COLLSYMBOL
5243         | COLLSYMBOL
5244         ;
5245     locale_name    : LOC_NAME
5246         | ' ' LOC_NAME ' '
5247         ;
5248     /* The following is the LC_CTYPE category grammar */
5249     lc_ctype       : ctype_hdr ctype_keywords      ctype_tlr
5250         | ctype_hdr 'copy' locale_name EOL ctype_tlr
5251         ;
5252     ctype_hdr      : 'LC_CTYPE' EOL
5253         ;
5254     ctype_keywords : ctype_keywords ctype_keyword
5255         | ctype_keyword
5256         ;
5257     ctype_keyword  : charclass_keyword charclass_list EOL
5258         | charconv_keyword charconv_list EOL
5259         | 'charclass' charclass_namelist EOL
5260         ;
5261     charclass_namelist : charclass_namelist ';' CHARCLASS
5262         | CHARCLASS
5263         ;
5264     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5265         | 'punct' | 'xdigit' | 'space' | 'print'
5266         | 'graph' | 'blank' | 'cntrl' | 'alnum'
5267         | CHARCLASS
5268         ;
5269     charclass_list  : charclass_list ';' char_symbol
5270         | charclass_list ';' ELLIPSIS ';' char_symbol
5271         | char_symbol
5272         ;
5273     charconv_keyword : 'toupper'
5274         | 'tolower'
5275         ;
5276     charconv_list   : charconv_list ';' charconv_entry
5277         | charconv_entry
5278         ;
5279     charconv_entry  : '(' char_symbol ',' char_symbol ')'
5280         ;
5281     ctype_tlr      : 'END' 'LC_CTYPE' EOL
5282         ;
5283     /* The following is the LC_COLLATE category grammar */
5284     lc_collate     : collate_hdr collate_keywords      collate_tlr

```

```

5285 | collate_hdr 'copy' locale_name EOL collate_tlr
5286 |
5287 collate_hdr : 'LC_COLLATE' EOL
5288 |
5289 collate_keywords : order_statements
5290 | opt_statements order_statements
5291 |
5292 opt_statements : opt_statements collating_symbols
5293 | opt_statements collating_elements
5294 | collating_symbols
5295 | collating_elements
5296 |
5297 collating_symbols : 'collating-symbol' COLLSYMBOL EOL
5298 |
5299 collating_elements : 'collating-element' COLLELEMENT
5300 | 'from' "" elem_list "" EOL
5301 |
5302 order_statements : order_start collation_order order_end
5303 |
5304 order_start : 'order_start' EOL
5305 | 'order_start' order_opts EOL
5306 |
5307 order_opts : order_opts ';' order_opt
5308 | order_opt
5309 |
5310 order_opt : order_opt ',' opt_word
5311 | opt_word
5312 |
5313 opt_word : 'forward' | 'backward' | 'position'
5314 |
5315 collation_order : collation_order collation_entry
5316 | collation_entry
5317 |
5318 collation_entry : COLLSYMBOL EOL
5319 | collation_element weight_list EOL
5320 | collation_element EOL
5321 |
5322 collation_element : char_symbol
5323 | COLLELEMENT
5324 | ELLIPSIS
5325 | 'UNDEFINED'
5326 |
5327 weight_list : weight_list ';' weight_symbol
5328 | weight_list ';'
5329 | weight_symbol
5330 |

```

## Locale

## Locale Definition Grammar

```

5331     weight_symbol      : /* empty */
5332                          | char_symbol
5333                          | COLLSYMBOL
5334                          | ''' elem_list '''
5335                          | ''' symb_list '''
5336                          | ELLIPSIS
5337                          | 'IGNORE'
5338                          ;
5339     order_end           : 'order_end' EOL
5340                          ;
5341     collate_tlr         : 'END' 'LC_COLLATE' EOL
5342                          ;
5343     /* The following is the LC_MESSAGES category grammar */
5344     lc_messages         : messages_hdr messages_keywords      messages_tlr
5345                          | messages_hdr 'copy' locale_name EOL messages_tlr
5346                          ;
5347     messages_hdr       : 'LC_MESSAGES' EOL
5348                          ;
5349     messages_keywords  : messages_keywords messages_keyword
5350                          | messages_keyword
5351                          ;
5352     messages_keyword   : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5353                          | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5354                          ;
5355     messages_tlr       : 'END' 'LC_MESSAGES' EOL
5356                          ;
5357     /* The following is the LC_MONETARY category grammar */
5358     lc_monetary         : monetary_hdr monetary_keywords      monetary_tlr
5359                          | monetary_hdr 'copy' locale_name EOL monetary_tlr
5360                          ;
5361     monetary_hdr       : 'LC_MONETARY' EOL
5362                          ;
5363     monetary_keywords  : monetary_keywords monetary_keyword
5364                          | monetary_keyword
5365                          ;
5366     monetary_keyword   : mon_keyword_string mon_string EOL
5367                          | mon_keyword_char NUMBER EOL
5368                          | mon_keyword_char '-1' EOL
5369                          | mon_keyword_grouping mon_group_list EOL
5370                          ;
5371     mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5372                          | 'mon_decimal_point' | 'mon_thousands_sep'
5373                          | 'positive_sign' | 'negative_sign'
5374                          ;
5375     mon_string         : ''' char_list '''

```

## Locale Definition Grammar

## Locale

```

5376         | '""'
5377         ;
5378     mon_keyword_char : 'int_frac_digits' | 'frac_digits'
5379         | 'p_cs_precedes' | 'p_sep_by_space'
5380         | 'n_cs_precedes' | 'n_sep_by_space'
5381         | 'p_sign_posn' | 'n_sign_posn'
5382         | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5383         | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5384         | 'int_p_sign_posn' | 'int_n_sign_posn'
5385         ;
5386     mon_keyword_grouping : 'mon_grouping'
5387         ;
5388     mon_group_list : NUMBER
5389         | mon_group_list ';' NUMBER
5390         ;
5391     monetary_tlr : 'END' 'LC_MONETARY' EOL
5392         ;
5393     /* The following is the LC_NUMERIC category grammar */
5394     lc_numeric : numeric_hdr numeric_keywords numeric_tlr
5395         | numeric_hdr 'copy' locale_name EOL numeric_tlr
5396         ;
5397     numeric_hdr : 'LC_NUMERIC' EOL
5398         ;
5399     numeric_keywords : numeric_keywords numeric_keyword
5400         | numeric_keyword
5401         ;
5402     numeric_keyword : num_keyword_string num_string EOL
5403         | num_keyword_grouping num_group_list EOL
5404         ;
5405     num_keyword_string : 'decimal_point'
5406         | 'thousands_sep'
5407         ;
5408     num_string : '"' char_list '"'
5409         | '""'
5410         ;
5411     num_keyword_grouping : 'grouping'
5412         ;
5413     num_group_list : NUMBER
5414         | num_group_list ';' NUMBER
5415         ;
5416     numeric_tlr : 'END' 'LC_NUMERIC' EOL
5417         ;
5418     /* The following is the LC_TIME category grammar */
5419     lc_time : time_hdr time_keywords time_tlr
5420         | time_hdr 'copy' locale_name EOL time_tlr

```

## Locale

## Locale Definition Grammar

```

5421                                     ;
5422     time_hdr                         : 'LC_TIME' EOL
5423                                     ;
5424     time_keywords                     : time_keywords time_keyword
5425     | time_keyword
5426                                     ;
5427     time_keyword                      : time_keyword_name time_list EOL
5428     | time_keyword_fmt time_string EOL
5429     | time_keyword_opt time_list EOL
5430                                     ;
5431     time_keyword_name                 : 'abday' | 'day' | 'abmon' | 'mon'
5432                                     ;
5433     time_keyword_fmt                 : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5434     | 'am_pm' | 't_fmt_ampm'
5435                                     ;
5436     time_keyword_opt                 : 'era' | 'era_d_fmt' | 'era_t_fmt'
5437     | 'era_d_t_fmt' | 'alt_digits'
5438                                     ;
5439     time_list                         : time_list ';' time_string
5440     | time_string
5441                                     ;
5442     time_string                      : '"' char_list '"'
5443                                     ;
5444     time_tlr                          : 'END' 'LC_TIME' EOL
5445                                     ;

```

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

5446

Chapter 8

5447

## Environment Variables

### 8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of POSIX.1-2008 for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to POSIX.1-2008, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of POSIX.1-2008 consist solely of uppercase letters, digits, and the <underscore> ('\_') from the characters defined in Table 6-1 (on page 125) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

**Note:** Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG\_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of POSIX.1-2008.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

*Environment Variable Definition*

*Environment Variables*

5481	ARFLAGS	IFS	MAILPATH	PS1
5482	CC	LANG	MAILRC	PS2
5483	CDPATH	LC_ALL	MAKEFLAGS	PS3
5484	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5485	CHARSET	LC_CTYPE	MANPATH	PWD
5486	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5487	DATMSK	LC_MONETARY	MORE	SECONDS
5488	DEAD	LC_NUMERIC	MSGVERB	SHELL
5489	EDITOR	LC_TIME	NLSPATH	TERM
5490	ENV	LDFLAGS	NPROC	TERMCAP
5491	EXINIT	LEX	OLDPWD	TERMINFO
5492	FC	LFLAGS	OPTARG	TMPDIR
5493	FCEDIT	LINENO	OPTERR	TZ
5494	FFLAGS	LINES	OPTIND	USER
5495	GET	LISTER	PAGER	VISUAL
5496	GFLAGS	LOGNAME	PATH	YACC
5497	HISTFILE	LPDEST	PPID	YFLAGS
5498	HISTORY	MAIL	PRINTER	
5499	HISTSZIE	MAILCHECK	PROCLANG	
5500	HOME	MAILER	PROJECTDIR	

5501 If the variables in the following two sections are present in the environment during the  
 5502 execution of an application or utility, they shall be given the meaning described below. Some are  
 5503 placed into the environment by the implementation at the time the user logs in; all can be added  
 5504 or changed by the user or any ancestor of the current process. The implementation adds or  
 5505 changes environment variables named in POSIX.1-2008 only as specified in POSIX.1-2008. If  
 5506 they are defined in the application's environment, the utilities in the Shell and Utilities volume  
 5507 of POSIX.1-2008 and the functions in the System Interfaces volume of POSIX.1-2008 assume they  
 5508 have the specified meaning. Conforming applications shall not set these environment variables  
 5509 to have meanings other than as described. See *getenv()* (on page 1008) and XCU Section 2.12 (on  
 5510 page 2331) for methods of accessing these variables.

5511 **8.2 Internationalization Variables**

5512 This section describes environment variables that are relevant to the operation of  
 5513 internationalized interfaces described in POSIX.1-2008.

5514 Users may use the following environment variables to announce specific localization  
 5515 requirements to applications. Applications can retrieve this information using the *setlocale()*  
 5516 function to initialize the correct behavior of the internationalized interfaces. The descriptions of  
 5517 the internationalization environment variables describe the resulting behavior only when the  
 5518 application locale is initialized in this way. The use of the internationalization variables by  
 5519 utilities described in the Shell and Utilities volume of POSIX.1-2008 is described in the  
 5520 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects  
 5521 described in this section.

5522 **LANG** This variable shall determine the locale category for native language, local  
 5523 customs, and coded character set in the absence of the *LC\_ALL* and other *LC\_\**  
 5524 (*LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*, *LC\_MONETARY*, *LC\_NUMERIC*,  
 5525 *LC\_TIME*) environment variables. This can be used by applications to  
 5526 determine the language to use for error messages and instructions, collating  
 5527 sequences, date formats, and so on.

## Environment Variables

## Internationalization Variables

5528	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_</i> ( <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> ) and the <i>LANG</i> environment variable.
5529		
5530		
5531		
5532		
5533	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5534		
5535		
5536		
5537		
5538	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5539		
5540		
5541		
5542		
5543		
5544		
5545	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i> function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-2008 should be affected by the setting of <i>LC_MESSAGES</i> .
5546		
5547		
5548		
5549		
5550		
5551		
5552		
5553	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.
5554		
5555		
5556	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the string conversion functions in <i>strtod()</i> . Additional semantics of this variable, if any, are implementation-defined.
5557		
5558		
5559		
5560		
5561	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strftime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5562		
5563		
5564	<i>NLSPATH</i>	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a filename, and an optional suffix.
5565		
5566		
5567		
5568		For example:
5569		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5570		defines that <i>catopen()</i> should look for all message catalogs in the directory <b>/system/nlslib</b> , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> ( <i>%N</i> ), with the suffix <b>.cat</b> .
5571		
5572		
5573		Conversion specifications consist of a ' <i>%</i> ' symbol, followed by a single-letter keyword. The following keywords are currently defined:
5574		

5575	%N	The value of the <i>name</i> parameter passed to <i>catopen</i> ().
5576	%L	The value of the <i>LC_MESSAGES</i> category.
5577	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5578	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5579	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5580	%%	A single ' % ' character.
5581		An empty string is substituted if the specified value is not currently defined.
5582		The separators <underscore> ('_') and <period> ('.') are not included in
5583		the %t and %c conversion specifications.
5584		Templates defined in <i>NLSPATH</i> are separated by <colon> characters (':'). A
5585		leading or two adjacent <colon> characters ("::") is equivalent to specifying
5586		%N. For example:
5587		<code>NLSPATH=":%N.cat:/nlslib/%L/%N.cat"</code>
5588		indicates to <i>catopen</i> () that it should look for the requested message catalog in
5589		<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the
5590		<i>LC_MESSAGES</i> category of the current locale.
5591		Users should not set the <i>NLSPATH</i> variable unless they have a specific reason
5592		to override the default system path. Setting <i>NLSPATH</i> to override the default
5593		system path produces undefined results in the standard utilities and in
5594		applications with appropriate privileges.
5595		The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5596		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5597		internationalized applications. The standard utilities shall make use of these environment
5598		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5599		for the utilities. If these variables specify locale categories that are not based upon the same
5600		underlying codeset, the results are unspecified.
5601		The values of locale categories shall be determined by a precedence order; the first condition met
5602		below determines the value:
5603	1.	If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall
5604		be used.
5605	2.	If the <i>LC_*</i> environment variable ( <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5606		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> ) is defined and is not null, the value of the
5607		environment variable shall be used to initialize the category that corresponds to the
5608		environment variable.
5609	3.	If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>
5610		environment variable shall be used.
5611	4.	If the <i>LANG</i> environment variable is not set or is set to the empty string, the
5612		implementation-defined default locale shall be used.
5613		If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
5614		behave in accordance with the rules in <a href="#">Section 7.2</a> (on page 136) for the associated category.
5615		If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was
5616		created in the output format used by the <i>localedef</i> utility; see OUTPUT FILES under <i>localedef</i> .
5617		Referencing such a pathname shall result in that locale being used for the indicated category.

5618 XSI If the locale value has the form:

5619 `language[_territory] [.codeset]`

5620 it refers to an implementation-provided locale, where settings of language, territory, and codeset  
5621 are implementation-defined.

5622 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are  
5623 defined to accept an additional field `@modifier`, which allows the user to select a specific instance  
5624 of localization data within a single category (for example, for selecting the dictionary as opposed  
5625 to the character ordering of data). The syntax for these environment variables is thus defined as:

5626 `[language[_territory] [.codeset] [@modifier]]`

5627 For example, if a user wanted to interact with the system in French, but required to sort German  
5628 text files, `LANG` and `LC_COLLATE` could be defined as:

5629 `LANG=Fr_FR`  
5630 `LC_COLLATE=De_DE`

5631 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for  
5632 example:

5633 `LC_COLLATE=De_DE@dict`

5634 An implementation may support other formats.

5635 If the locale value is not recognized by the implementation, the behavior is unspecified.

5636 At runtime, these values are bound to the locale of a process by calling the `setlocale()` function.

5637 Additional criteria for determining a valid locale name are implementation-defined.

### 5638 8.3 Other Environment Variables

5639 `COLUMNS` This variable shall represent a decimal integer >0 used to indicate the user's  
5640 preferred width in column positions for the terminal screen or window; see  
5641 Section 3.103 (on page 50). If this variable is unset or null, the implementation  
5642 determines the number of columns, appropriate for the terminal or window,  
5643 in an unspecified manner. When `COLUMNS` is set, any terminal-width  
5644 information implied by `TERM` is overridden. Users and conforming  
5645 applications should not set `COLUMNS` unless they wish to override the  
5646 system selection and produce output unrelated to the terminal characteristics.

5647 Users should not need to set this variable in the environment unless there is a  
5648 specific reason to override the implementation's default behavior, such as to  
5649 display data in an area arbitrarily smaller than the terminal or window.

5650 XSI `DATMSK` Indicates the pathname of the template file used by `getdate()`.

5651 `HOME` The system shall initialize this variable at the time of login to be a pathname of  
5652 the user's home directory. See `<pwd.h>`.

5653 `LINES` This variable shall represent a decimal integer >0 used to indicate the user's  
5654 preferred number of lines on a page or the vertical screen or window size in  
5655 lines. A line in this case is a vertical measure large enough to hold the tallest  
5656 character in the character set being displayed. If this variable is unset or null,  
5657 the implementation determines the number of lines, appropriate for the

5658		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5659		
5660		
5661		
5662		
5663		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5664		
5665		
5666	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See <a href="#">&lt;pwd.h&gt;</a> . For a value of <i>LOGNAME</i> to be portable across implementations of POSIX.1-2008, the value should be composed of characters from the portable filename character set.
5667		
5668		
5669		
5670	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
5671		
5672	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a <colon> (' : '). When a non-zero-length prefix is applied to this filename, a <slash> shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent <colon> characters (" : : "), as an initial <colon> preceding the rest of the list, or as a trailing <colon> following the rest of the list. A strictly conforming application shall use an actual pathname (such as <i>.</i> ) to represent the current working directory in <i>PATH</i> . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a <slash>, the search through the path prefixes shall not be performed. If the pathname begins with a <slash>, the specified path is resolved (see <a href="#">Section 4.12</a> , on page 111). If <i>PATH</i> is unset or is set to null, the path search is implementation-defined.
5673		
5674		
5675		
5676		
5677		
5678		
5679		
5680		
5681		
5682		
5683		
5684		
5685		
5686		
5687		
5688	<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any components that are dot or dot-dot. The value is set by the <i>cd</i> utility, and by the <i>sh</i> utility during initialization.
5689		
5690		
5691	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in XCU <a href="#">Chapter 2</a> (on page 2297), utilities may behave differently from those described in POSIX.1-2008.
5692		
5693		
5694		
5695	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
5696		
5697	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
5698		
5699		
5700		
5701	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>ctime_r()</i> , <i>localtime()</i> , <i>localtime_r()</i> , <i>strftime()</i> , <i>mktime()</i> , functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted for clarity):
5702		
5703		
5704		
5705		

## Environment Variables

## Other Environment Variables

5706 : *characters*

5707 or:

5708 *std offset dst offset, rule*

5709 If *TZ* is of the first format (that is, if the first character is a <colon>), the  
5710 characters following the <colon> are handled in an implementation-defined  
5711 manner.

5712 The expanded format (for all *TZ*s whose value does not have a <colon> as the  
5713 first character) is as follows:

5714 *stdoffset* [*dst* [*offset*] [, *start* [/time], *end* [/time]]]

5715 Where:

5716 *std* and *dst* Indicate no less than three, nor more than {TZNAME\_MAX},  
5717 bytes that are the designation for the standard (*std*) or the  
5718 alternative (*dst*—such as Daylight Savings Time) timezone. Only  
5719 *std* is required; if *dst* is missing, then the alternative time does  
5720 not apply in this locale.

5721 Each of these fields may occur in either of two formats quoted or  
5722 unquoted:

5723 — In the quoted form, the first character shall be the <less-  
5724 than-sign> ('<') character and the last character shall be  
5725 the <greater-than-sign> ('>') character. All characters  
5726 between these quoting characters shall be alphanumeric  
5727 characters from the portable character set in the current  
5728 locale, the <plus-sign> ('+') character, or the minus-sign  
5729 ('-') character. The *std* and *dst* fields in this case shall not  
5730 include the quoting characters.

5731 — In the unquoted form, all characters in these fields shall be  
5732 alphabetic characters from the portable character set in the  
5733 current locale.

5734 The interpretation of these fields is unspecified if either field is  
5735 less than three bytes (except for the case when *dst* is missing),  
5736 more than {TZNAME\_MAX} bytes, or if they contain characters  
5737 other than those specified.

5738 *offset* Indicates the value added to the local time to arrive at  
5739 Coordinated Universal Time. The *offset* has the form:

5740 *hh* [: *mm* [: *ss* ]]

5741 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)  
5742 shall be required and may be a single digit. The *offset* following  
5743 *std* shall be required. If no *offset* follows *dst*, the alternative time  
5744 is assumed to be one hour ahead of standard time. One or more  
5745 digits may be used; the value is always interpreted as a decimal  
5746 number. The hour shall be between zero and 24, and the minutes  
5747 (and seconds)—if present—between zero and 59. The result of  
5748 using values outside of this range is unspecified. If preceded by  
5749 a '-', the timezone shall be east of the Prime Meridian;  
5750 otherwise, it shall be west (which may be indicated by an  
5751 optional preceding '+').

5752 5753 5754 5755 5756 5757 5758 5759 5760 5761 5762 5763 5764 5765 5766 5767 5768 5769 5770 5771 5772 5773 5774	<i>rule</i>	<p>Indicates when to change to and back from the alternative time. The <i>rule</i> has the form:</p> <p><i>date</i>[/<i>time</i>], <i>date</i>[/<i>time</i>]</p> <p>where the first <i>date</i> describes when the change from standard to alternative time occurs and the second <i>date</i> describes when the change back happens. Each <i>time</i> field describes when, in current local time, the change to the other time is made.</p> <p>The format of <i>date</i> is one of the following:</p> <p><i>Jn</i> The Julian day <i>n</i> (<math>1 \leq n \leq 365</math>). Leap days shall not be counted. That is, in all years—including leap years—February 28 is day 59 and March 1 is day 60. It is impossible to refer explicitly to the occasional February 29.</p> <p><i>n</i> The zero-based Julian day (<math>0 \leq n \leq 365</math>). Leap days shall be counted, and it is possible to refer to February 29.</p> <p><i>Mm.n.d</i> The <i>d</i>'th day (<math>0 \leq d \leq 6</math>) of week <i>n</i> of month <i>m</i> of the year (<math>1 \leq n \leq 5</math>, <math>1 \leq m \leq 12</math>, where week 5 means “the last <i>d</i> day in month <i>m</i>” which may occur in either the fourth or the fifth week). Week 1 is the first week in which the <i>d</i>'th day occurs. Day zero is Sunday.</p> <p>The <i>time</i> has the same format as <i>offset</i> except that no leading sign ('-' or '+') is allowed. The default, if <i>time</i> is not given, shall be 02:00:00.</p>
--	-------------	---

5775

Chapter 9

5776

## Regular Expressions

5777  
5778

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

5779  
5780  
5781  
5782  
5783

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

5784  
5785  
5786  
5787  
5788  
5789

The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 183) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in Section 9.4 (on page 188); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of POSIX.1-2008 under *regcomp()*, *regex()*, and related functions.

### 9.1 Regular Expression Definitions

5790

For the purposes of this section, the following definitions shall apply:

5791

#### entire regular expression

5792  
5793  
5794

The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

5795

#### matched

5796  
5797  
5798

A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

5799  
5800  
5801  
5802  
5803  
5804

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

5805  
5806  
5807  
5808  
5809  
5810  
5811

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where "first" is defined to mean "begins earliest in the string". If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example, the BRE "bb\*" matches the second to fourth characters of the string "abbbc", and the ERE "(wee|week)(knights|night)" matches all ten characters of the string "weeknights".

5812  
5813  
5814

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE

5815            "\ (. \* \) . \*" against "abcdef", the subexpression " (\1) " is "abcdef", and matching  
5816            the BRE "\ (a \* \) \*" against "bc", the subexpression " (\1) " is the null string.

5817            When a multi-character collating element in a bracket expression (see Section 9.3.5, on page  
5818            184) is involved, the longest sequence shall be measured in characters consumed from the  
5819            string to be matched; that is, the collating element counts not as one element, but as the  
5820            number of characters it matches.

#### 5821            **BRE (ERE) matching a single character**

5822            A BRE or ERE that shall match either a single character or a single collating element.

5823            Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5, on page  
5824            184) can match a collating element.

#### 5825            **BRE (ERE) matching multiple characters**

5826            A BRE or ERE that shall match a concatenation of single characters or collating elements.

5827            Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE  
5828            (ERE) special characters.

#### 5829            **invalid**

5830            This section uses the term “invalid” for certain constructs or conditions. Invalid REs shall  
5831            cause the utility or function using the RE to generate an error condition. When invalid is not  
5832            used, violations of the specified syntax or semantics for REs produce undefined results: this  
5833            may entail an error, enabling an extended syntax for that RE, or using the construct in error  
5834            as literal characters to be matched. For example, the BRE construct "\ {1, 2, 3 \}" does not  
5835            comply with the grammar. A conforming application cannot rely on it producing an error  
5836            nor matching the literal characters "\ {1, 2, 3 \}".

## 5837            **9.2 Regular Expression General Requirements**

5838            The requirements in this section shall apply to both basic and extended regular expressions.

5839            The use of regular expressions is generally associated with text processing. REs (BREs and EREs)  
5840            operate on text strings; that is, zero or more characters followed by an end-of-string delimiter  
5841            (typically NUL). Some utilities employing regular expressions limit the processing to lines; that  
5842            is, zero or more characters followed by a <newline>. In the regular expression processing  
5843            described in POSIX.1-2008, the <newline> is regarded as an ordinary character and both a  
5844            <period> and a non-matching list can match one. The Shell and Utilities volume of  
5845            POSIX.1-2008 specifies within the individual descriptions of those standard utilities employing  
5846            regular expressions whether they permit matching of <newline> characters; if not stated  
5847            otherwise, the use of literal <newline> characters or any escape sequence equivalent produces  
5848            undefined results. Those utilities (like *grep*) that do not allow <newline> characters to match are  
5849            responsible for eliminating any <newline> from strings before matching against the RE. The  
5850            *regcomp()* function in the System Interfaces volume of POSIX.1-2008, however, can provide  
5851            support for such processing without violating the rules of this section.

5852            The interfaces specified in POSIX.1-2008 do not permit the inclusion of a NUL character in an RE  
5853            or in the string to be matched. If during the operation of a standard utility a NUL is included in  
5854            the text designated to be matched, that NUL may designate the end of the text string for the  
5855            purposes of matching.

5856            When a standard utility or function that uses regular expressions specifies that pattern matching  
5857            shall be performed without regard to the case (uppercase or lowercase) of either data or  
5858            patterns, then when each character in the string is matched against the pattern, not only the



- 5892 — As the first character of a subexpression (after an initial '<sup>^</sup>', if any); see [Section](#)  
5893 [9.3.6](#) (on page 186)
- 5894 <sup>^</sup> The <circumflex> shall be special when used as:  
5895 — An anchor (see [Section 9.3.8](#), on page 187)  
5896 — The first character of a bracket expression (see [Section 9.3.5](#))
- 5897 \$ The <dollar-sign> shall be special when used as an anchor.

### 5898 9.3.4 Periods in BREs

5899 A <period> ('.'), when used outside a bracket expression, is a BRE that shall match any  
5900 character in the supported character set except NUL.

### 5901 9.3.5 RE Bracket Expression

5902 A bracket expression (an expression enclosed in square brackets, "[ ]") is an RE that shall  
5903 match a single collating element contained in the non-empty set of collating elements  
5904 represented by the bracket expression.

5905 The following rules and definitions apply to bracket expressions:

- 5906 1. A bracket expression is either a matching list expression or a non-matching list  
5907 expression. It consists of one or more expressions: collating elements, collating symbols,  
5908 equivalence classes, character classes, or range expressions. The <right-square-bracket>  
5909 (' ] ') shall lose its special meaning and represent itself in a bracket expression if it occurs  
5910 first in the list (after an initial <circumflex> (' ^ '), if any). Otherwise, it shall terminate  
5911 the bracket expression, unless it appears in a collating symbol (such as "[ . ] . ") or is the  
5912 ending <right-square-bracket> for a collating symbol, equivalence class, or character  
5913 class. The special characters '<period>', '<asterisk>', '<left-square-bracket>', and '<backslash>'  
5914 (<period>, <asterisk>, <left-square-bracket>, and <backslash>, respectively) shall lose their special meaning within a bracket  
5915 expression.  
  
5916 The character sequences "[ . ", "[ = ", and "[ : " (<left-square-bracket> followed by a  
5917 <period>, <equals-sign>, or <colon>) shall be special inside a bracket expression and are  
5918 used to delimit collating symbols, equivalence class expressions, and character class  
5919 expressions. These symbols shall be followed by a valid expression and the matching  
5920 terminating sequence ". ] ", "= ] ", or ": ] ", as described in the following items.
- 5921 2. A matching list expression specifies a list that shall match any single-character collating  
5922 element in any of the expressions represented in the list. The first character in the list shall  
5923 not be the <circumflex>; for example, "[abc]" is an RE that matches any of the  
5924 characters 'a', 'b', or 'c'. It is unspecified whether a matching list expression matches  
5925 a multi-character collating element that is matched by one of the expressions.
- 5926 3. A non-matching list expression begins with a <circumflex> (' ^ '), and specifies a list that  
5927 shall match any single-character collating element except for the expressions represented  
5928 in the list after the leading <circumflex>. For example, "[^abc]" is an RE that matches  
5929 any character except the characters 'a', 'b', or 'c'. It is unspecified whether a non-  
5930 matching list expression matches a multi-character collating element that is not matched  
5931 by any of the expressions. The <circumflex> shall have this special meaning only when it  
5932 occurs first in the list, immediately following the <left-square-bracket>.

5933 4. A collating symbol is a collating element enclosed within bracket-period ("[" and  
 5934 ". ]") delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page  
 5935 149). Conforming applications shall represent multi-character collating elements as  
 5936 collating symbols when it is necessary to distinguish them from a list of the individual  
 5937 characters that make up the multi-character collating element. For example, if the string  
 5938 "ch" is a collating element defined using the line:

```
5939 collating-element <ch-digraph> from "<c><h>"
```

5940 in the locale definition, the expression "[.ch.]" shall be treated as an RE containing  
 5941 the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'.  
 5942 Collating symbols are recognized only inside bracket expressions. If the string is not a  
 5943 collating element in the current locale, the expression is invalid.

5944 5. An equivalence class expression shall represent the set of collating elements belonging to  
 5945 an equivalence class, as described in Section 7.3.2.4 (on page 149). Only primary  
 5946 equivalence classes shall be recognized. The class shall be expressed by enclosing any one  
 5947 of the collating elements in the equivalence class within bracket-equal ("[" and "=")  
 5948 delimiters. For example, if 'a', 'à', and '^' belong to the same equivalence class, then  
 5949 "[[=a=b]", "[[=à=b]", and "[[=^=b]" are each equivalent to "[aà^b]". If the  
 5950 collating element does not belong to an equivalence class, the equivalence class  
 5951 expression shall be treated as a collating symbol.

5952 6. A character class expression shall represent the union of two sets:

5953 a. The set of single-character collating elements whose characters belong to the  
 5954 character class, as defined in the LC\_CTYPE category in the current locale.

5955 b. An unspecified set of multi-character collating elements.

5956 All character classes specified in the current locale shall be recognized. A character class  
 5957 expression is expressed as a character class name enclosed within bracket-<colon> ("[:"  
 5958 and ":]") delimiters.

5959 The following character class expressions shall be supported in all locales:

```
5960 [:alnum:]    [:cntrl:]    [:lower:]    [:space:]
5961 [:alpha:]    [:digit:]    [:print:]    [:upper:]
5962 [:blank:]    [:graph:]    [:punct:]    [:xdigit:]
```

5963 In addition, character class expressions of the form:

```
5964 [:name:]
```

5965 are recognized in those locales where the *name* keyword has been given a **charclass**  
 5966 definition in the LC\_CTYPE category.

5967 7. In the POSIX locale, a range expression represents the set of collating elements that fall  
 5968 between two elements in the collation sequence, inclusive. In other locales, a range  
 5969 expression has unspecified behavior: strictly conforming applications shall not rely on  
 5970 whether the range expression is valid, or on the set of collating elements matched. A  
 5971 range expression shall be expressed as the starting point and the ending point separated  
 5972 by a <hyphen> ('-').

5973 In the following, all examples assume the POSIX locale.

5974 The starting range point and the ending range point shall be a collating element or  
 5975 collating symbol. An equivalence class expression used as a starting or ending point of a  
 5976 range expression produces unspecified results. An equivalence class can be used portably  
 5977 within a bracket expression, but only outside the range. If the represented set of collating

- 5978 elements is empty, it is unspecified whether the expression matches nothing, or is treated  
5979 as invalid.
- 5980 The interpretation of range expressions where the ending range point is also the starting  
5981 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.
- 5982 The <hyphen> character shall be treated as itself if it occurs first (after an initial '^', if  
5983 any) or last in the list, or as an ending range point in a range expression. As examples, the  
5984 expressions "[ac]" and "[ac-]" are equivalent and match any of the characters 'a',  
5985 'c', or '-'; "[^ac]" and "[^ac-]" are equivalent and match any characters except  
5986 'a', 'c', or '-'; the expression "[%--]" matches any of the characters between '%'  
5987 and '-' inclusive; the expression "[--@]" matches any of the characters between '-'  
5988 and '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@',  
5989 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a <hyphen> as  
5990 the starting range point, it shall either come first in the bracket expression or be specified  
5991 as a collating symbol; for example, "[][.-.]0]", which matches either a <right-  
5992 square-bracket> or any character or collating element that collates between <hyphen>  
5993 and 0, inclusive.
- 5994 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the  
5995 '^', if any) and the '-' last within the bracket expression.

### 5996 9.3.6 BREs Matching Multiple Characters

- 5997 The following rules can be used to construct BREs matching multiple characters from BREs  
5998 matching a single character:
- 5999 1. The concatenation of BREs shall match the concatenation of the strings matched by each  
6000 component of the BRE.
  - 6001 2. A subexpression can be defined within a BRE by enclosing it between the character pairs  
6002 "\" and "\". Such a subexpression shall match whatever it would have matched  
6003 without the "\" and "\"), except that anchoring within subexpressions is optional  
6004 behavior; see Section 9.3.8 (on page 187). Subexpressions can be arbitrarily nested.
  - 6005 3. The back-reference expression '\n' shall match the same (possibly empty) string of  
6006 characters as was matched by a subexpression enclosed between "\" and "\"  
6007 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the  
6008 *n*th subexpression (the one that begins with the *n*th "\" from the beginning of the  
6009 pattern and ends with the corresponding paired "\"). The expression is invalid if less  
6010 than *n* subexpressions precede the '\n'. The string matched by a contained  
6011 subexpression shall be within the string matched by the containing subexpression. If the  
6012 containing subexpression does not match, or if there is no match for the contained  
6013 subexpression within the string matched by the containing subexpression, then back-  
6014 reference expressions corresponding to the contained subexpression shall not match.  
6015 When a subexpression matches more than one string, a back-reference expression  
6016 corresponding to the subexpression shall refer to the last matched string. For example, the  
6017 expression "^ (.\*) \\1\$" matches lines consisting of two adjacent appearances of the  
6018 same string, and the expression "(a) \* \\1" fails to match 'a', the expression  
6019 "(a(b) \*) \* \\2" fails to match 'abab', and the expression "^ (ab\*) \* \\1\$" matches  
6020 'ababbabb', but fails to match 'ababbab'.
  - 6021 4. When a BRE matching a single character, a subexpression, or a back-reference is followed  
6022 by the special character <asterisk> ('\*'), together with that <asterisk> it shall match  
6023 what zero or more consecutive occurrences of the BRE would match. For example,  
6024 "[ab] \*" and "[ab] [ab]" are equivalent when matching the string "ab".

6025 5. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6026 by an interval expression of the format " $\{m\}$ ", " $\{m,\}$ ", or " $\{m,n\}$ ", together  
 6027 with that interval expression it shall match what repeated consecutive occurrences of the  
 6028 BRE would match. The values of  $m$  and  $n$  are decimal integers in the range  $0$   
 6029  $\leq m \leq n \leq \{RE\_DUP\_MAX\}$ , where  $m$  specifies the exact or minimum number of occurrences  
 6030 and  $n$  specifies the maximum number of occurrences. The expression " $\{m\}$ " shall  
 6031 match exactly  $m$  occurrences of the preceding BRE, " $\{m,\}$ " shall match at least  $m$   
 6032 occurrences, and " $\{m,n\}$ " shall match any number of occurrences between  $m$  and  $n$   
 6033 inclusive.

6034 For example, in the string "abababcccccd" the BRE " $c\{3\}$ " is matched by  
 6035 characters seven to nine, the BRE " $\{m,\}$ " is not matched at all, and the BRE  
 6036 " $c\{1,3\}d$ " is matched by characters ten to thirteen.

6037 The behavior of multiple adjacent duplication symbols ('\*' and intervals) produces undefined  
 6038 results.

6039 A subexpression repeated by an <asterisk> ('\*') or an interval expression shall not match a null  
 6040 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or  
 6041 minimum number of occurrences for the interval expression.

### 6042 9.3.7 BRE Precedence

6043 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[ ]
Subexpressions/back-references	\( \) \n
Single-character-BRE duplication	* \{m,n\}
Concatenation	
Anchoring	^ \$

### 6052 9.3.8 BRE Expression Anchoring

6053 A BRE can be limited to matching strings that begin or end a line; this is called "anchoring".  
 6054 The <circumflex> and <dollar-sign> special characters shall be considered BRE anchors in the  
 6055 following contexts:

6056 1. A <circumflex> ('^') shall be an anchor when used as the first character of an entire BRE.  
 6057 The implementation may treat the <circumflex> as an anchor when used as the first  
 6058 character of a subexpression. The <circumflex> shall anchor the expression (or optionally  
 6059 subexpression) to the beginning of a string; only sequences starting at the first character  
 6060 of a string shall be matched by the BRE. For example, the BRE " $^ab$ " matches "ab" in  
 6061 the string "abcdef", but fails to match in the string "cdefab". The BRE " $\{m,\}$ "  
 6062 may match the former string. A portable BRE shall escape a leading <circumflex> in a  
 6063 subexpression to match a literal circumflex.

6064 2. A <dollar-sign> ('\$') shall be an anchor when used as the last character of an entire BRE.  
 6065 The implementation may treat a <dollar-sign> as an anchor when used as the last  
 6066 character of a subexpression. The <dollar-sign> shall anchor the expression (or optionally  
 6067 subexpression) to the end of the string being matched; the <dollar-sign> can be said to  
 6068 match the end-of-string following the last character.

- 6069 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the  
6070 BRE "^abcdef\$" matches strings consisting only of "abcdef".

## 6071 9.4 Extended Regular Expressions

6072 The extended regular expression (ERE) notation and construction rules shall apply to utilities  
6073 defined as using extended regular expressions; any exceptions to the following rules are noted  
6074 in the descriptions of the specific utilities using EREs.

### 6075 9.4.1 EREs Matching a Single Character or Collating Element

6076 An ERE ordinary character, a special character preceded by a <backslash> or a <period> shall  
6077 match a single character. A bracket expression shall match a single character or a single collating  
6078 element. An ERE matching a single character enclosed in parentheses shall match the same as  
6079 the ERE without parentheses would have matched.

### 6080 9.4.2 ERE Ordinary Characters

6081 An ordinary character is an ERE that matches itself. An ordinary character is any character in the  
6082 supported character set, except for the ERE special characters listed in Section 9.4.3. The  
6083 interpretation of an ordinary character preceded by a <backslash> ('\\') is undefined.

### 6084 9.4.3 ERE Special Characters

6085 An ERE special character has special properties in certain contexts. Outside those contexts, or  
6086 when preceded by a <backslash>, such a character shall be an ERE that matches the special  
6087 character itself. The extended regular expression special characters and the contexts in which  
6088 they shall have their special meaning are as follows:

- 6089 . [ \ ( The <period>, <left-square-bracket>, <backslash>, and <left-parenthesis> shall be  
6090 special except when used in a bracket expression (see Section 9.3.5, on page 184).  
6091 Outside a bracket expression, a <left-parenthesis> immediately followed by a <right-  
6092 parenthesis> produces undefined results.
- 6093 ) The <right-parenthesis> shall be special when matched with a preceding <left-  
6094 parenthesis>, both outside a bracket expression.
- 6095 \* + ? | The <asterisk>, <plus-sign>, <question-mark>, and <left-brace> shall be special except  
6096 when used in a bracket expression (see Section 9.3.5, on page 184). Any of the  
6097 following uses produce undefined results:
- 6098 — If these characters appear first in an ERE, or immediately following a <vertical-  
6099 line>, <circumflex>, or <left-parenthesis>
  - 6100 — If a <left-brace> is not part of a valid interval expression (see Section 9.4.6, on  
6101 page 189)
- 6102 | The <vertical-line> is special except when used in a bracket expression (see Section  
6103 9.3.5, on page 184). A <vertical-line> appearing first or last in an ERE, or immediately  
6104 following a <vertical-line> or a <left-parenthesis>, or immediately preceding a <right-  
6105 parenthesis>, produces undefined results.

- 6106            $\wedge$        The <circumflex> shall be special when used as:
- 6107                     — An anchor (see [Section 9.4.9](#), on page 190)
- 6108                     — The first character of a bracket expression (see [Section 9.3.5](#), on page 184)
- 6109            $\$$        The <dollar-sign> shall be special when used as an anchor.

#### 6110 9.4.4 Periods in EREs

6111           A <period> ('.'), when used outside a bracket expression, is an ERE that shall match any  
6112           character in the supported character set except NUL.

#### 6113 9.4.5 ERE Bracket Expression

6114           The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see [Section](#)  
6115           [9.3.5](#) (on page 184).

#### 6116 9.4.6 EREs Matching Multiple Characters

6117           The following rules shall be used to construct EREs matching multiple characters from EREs  
6118           matching a single character:

- 6119           1. A concatenation of EREs shall match the concatenation of the character sequences  
6120           matched by each component of the ERE. A concatenation of EREs enclosed in parentheses  
6121           shall match whatever the concatenation without the parentheses matches. For example,  
6122           both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of  
6123           the string "abcdefabcdef".
- 6124           2. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6125           by the special character <plus-sign> ('+'), together with that <plus-sign> it shall match  
6126           what one or more consecutive occurrences of the ERE would match. For example, the  
6127           ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde".  
6128           And, "[ab]+" and "[ab][ab]\*" are equivalent.
- 6129           3. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6130           by the special character <asterisk> ('\*'), together with that <asterisk> it shall match  
6131           what zero or more consecutive occurrences of the ERE would match. For example, the  
6132           ERE "b\*c" matches the first character in the string "cabbbbcde", and the ERE "b\*cd"  
6133           matches the third to seventh characters in the string "cabbbbcdebbbbbbbcdbc". And,  
6134           "[ab]\*" and "[ab][ab]" are equivalent when matching the string "ab".
- 6135           4. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6136           by the special character <question-mark> ('?'), together with that <question-mark> it  
6137           shall match what zero or one consecutive occurrences of the ERE would match. For  
6138           example, the ERE "b?c" matches the second character in the string "acabbbbcde".
- 6139           5. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6140           by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that  
6141           interval expression it shall match what repeated consecutive occurrences of the ERE  
6142           would match. The values of *m* and *n* are decimal integers in the range 0  
6143            $\leq m \leq n \leq \{RE\_DUP\_MAX\}$ , where *m* specifies the exact or minimum number of occurrences  
6144           and *n* specifies the maximum number of occurrences. The expression "{m}" matches  
6145           exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and  
6146           "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.

6147 For example, in the string "abababcccccccd" the ERE "c{3}" is matched by characters  
 6148 seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6149 The behavior of multiple adjacent duplication symbols ('+', '\*', '?', and intervals) produces  
 6150 undefined results.

6151 An ERE matching a single character repeated by an '\*', '?', or an interval expression shall not  
 6152 match a null expression unless this is the only match for the repetition or it is necessary to satisfy  
 6153 the exact or minimum number of occurrences for the interval expression.

6154 **9.4.7 ERE Alternation**

6155 Two EREs separated by the special character <vertical-line> ('|') shall match a string that is  
 6156 matched by either. For example, the ERE "a(bc|d)" matches the string "abc" and the  
 6157 string "ad". Single characters, or expressions matching single characters, separated by the  
 6158 <vertical-line> and enclosed in parentheses, shall be treated as an ERE matching a single  
 6159 character.

6160 **9.4.8 ERE Precedence**

6161 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Grouping	()
Single-character-ERE duplication	* + ? {m,n}
Concatenation	
Anchoring	^ \$
Alternation	

6171 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"  
 6172 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of  
 6173 precedence than alternation).

6174 **9.4.9 ERE Expression Anchoring**

6175 An ERE can be limited to matching strings that begin or end a line; this is called "anchoring".  
 6176 The <circumflex> and <dollar-sign> special characters shall be considered ERE anchors when  
 6177 used anywhere outside a bracket expression. This shall have the following effects:

- 6178 1. A <circumflex> ('^') outside a bracket expression shall anchor the expression or  
 6179 subexpression it begins to the beginning of a string; such an expression or subexpression  
 6180 can match only a sequence starting at the first character of a string. For example, the EREs  
 6181 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string  
 6182 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents  
 6183 the expression "^b" from matching starting at the first character.
- 6184 2. A <dollar-sign> ('\$') outside a bracket expression shall anchor the expression or  
 6185 subexpression it ends to the end of a string; such an expression or subexpression can  
 6186 match only a sequence ending at the last character of a string. For example, the EREs  
 6187 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string

6188 "cdefab", and the ERE "e\$*f*" is valid, but can never match because the '*f*' prevents  
6189 the expression "e\$" from matching ending at the last character.

## 6190 9.5 Regular Expression Grammar

6191 Grammars describing the syntax of both basic and extended regular expressions are presented in  
6192 this section. The grammar takes precedence over the text. See XCU [Section 1.3](#) (on page 2287).

### 6193 9.5.1 BRE/ERE Grammar Lexical Conventions

6194 The lexical conventions for regular expressions are as described in this section.

6195 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6196 The following tokens are processed (in addition to those string constants shown in the  
6197 grammar):

6198 **COLL\_ELEM\_SINGLE** Any single-character collating element, unless it is a **META\_CHAR**.

6199 **COLL\_ELEM\_MULTI** Any multi-character collating element.

6200 **BACKREF** Applicable only to basic regular expressions. The character string  
6201 consisting of a <backslash> character followed by a single-digit  
6202 numeral, '1' to '9'.

6203 **DUP\_COUNT** Represents a numeric constant. It shall be an integer in the range 0  
6204 ≤**DUP\_COUNT** ≤**RE\_DUP\_MAX**. This token is only recognized  
6205 when the context of the grammar requires it. At all other times, digits  
6206 not preceded by a <backslash> character are treated as **ORD\_CHAR**.

6207 **META\_CHAR** One of the characters:

6208 **^** When found first in a bracket expression

6209 **-** When found anywhere but first (after an initial '**^**', if any)  
6210 or last in a bracket expression, or as the ending range point  
6211 in a range expression

6212 **]** When found anywhere but first (after an initial '**^**', if any)  
6213 in a bracket expression

6214 **L\_ANCHOR** Applicable only to basic regular expressions. The character '**^**'  
6215 when it appears as the first character of a basic regular expression  
6216 and when not **QUOTED\_CHAR**. The '**^**' may be recognized as an  
6217 anchor elsewhere; see [Section 9.3.8](#) (on page 187).

6218 **ORD\_CHAR** A character, other than one of the special characters in **SPEC\_CHAR**.

6219 **QUOTED\_CHAR** In a BRE, one of the character sequences:

6220 `\^ \. \* \[ \ $ \ \`

6221 In an ERE, one of the character sequences:

6222 `\^ \. \[ \ $ \ ( \ ) \ |`

6223 `\* \+ \? \{ \ \`

6224	<b>R_ANCHOR</b>	(Applicable only to basic regular expressions.) The character '\$' when it appears as the last character of a basic regular expression and when not <b>QUOTED_CHAR</b> . The '\$' may be recognized as an anchor elsewhere; see <a href="#">Section 9.3.8</a> (on page 187).
6225		
6226		
6227		
6228	<b>SPEC_CHAR</b>	For basic regular expressions, one of the following special characters:
6229	.	Anywhere outside bracket expressions
6230	\	Anywhere outside bracket expressions
6231	[	Anywhere outside bracket expressions
6232	^	When used as an anchor (see <a href="#">Section 9.3.8</a> , on page 187) or when first in a bracket expression
6233		
6234	\$	When used as an anchor
6235	*	Anywhere except first in an entire RE, anywhere in a bracket expression, directly following "\(", directly following an anchoring '^'
6236		
6237		
6238		For extended regular expressions, shall be one of the following special characters found anywhere outside bracket expressions:
6239		
6240	^	.
6241	*	+ ? { }
6242		(The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.)
6243		

## 6244 9.5.2 RE and Bracket Expression Grammar

6245 This section presents the grammar for basic regular expressions, including the bracket  
6246 expression grammar that is common to both BREs and EREs.

```

6247 %token   ORD_CHAR QUOTED_CHAR DUP_COUNT
6248 %token   BACKREF L_ANCHOR R_ANCHOR
6249 %token   Back_open_paren  Back_close_paren
6250 /*      '\('              '\)'          */
6251 %token   Back_open_brace  Back_close_brace
6252 /*      '\{'              '\}'          */
6253 /* The following tokens are for the Bracket Expression
6254    grammar common to both REs and EREs. */
6255 %token   COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6256 %token   Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6257 /*      '['              '='          '['      '.'      ':'          ':'      */
6258 %token   class_name
6259 /* class_name is a keyword to the LC_CTYPE locale category */
6260 /* (representing a character class) in the current locale */
6261 /* and is only recognized between [: and :] */
6262 %start   basic_reg_exp
6263 %%

```

## Regular Expressions

## Regular Expression Grammar

```

6264      /* -----
6265         Basic Regular Expression
6266         -----
6267      */
6268      basic_reg_exp  :      RE_expression
6269                    | L_ANCHOR
6270                    |                                R_ANCHOR
6271                    | L_ANCHOR                    R_ANCHOR
6272                    | L_ANCHOR RE_expression
6273                    |                                RE_expression R_ANCHOR
6274                    | L_ANCHOR RE_expression R_ANCHOR
6275                    ;
6276      RE_expression :      simple_RE
6277                    | RE_expression simple_RE
6278                    ;
6279      simple_RE    : nondupl_RE
6280                    | nondupl_RE RE_dupl_symbol
6281                    ;
6282      nondupl_RE   : one_char_or_coll_elem_RE
6283                    | Back_open_paren RE_expression Back_close_paren
6284                    | BACKREF
6285                    ;
6286      one_char_or_coll_elem_RE : ORD_CHAR
6287                    | QUOTED_CHAR
6288                    | '.'
6289                    | bracket_expression
6290                    ;
6291      RE_dupl_symbol : '*'
6292                    | Back_open_brace DUP_COUNT                    Back_close_brace
6293                    | Back_open_brace DUP_COUNT ','                    Back_close_brace
6294                    | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6295                    ;
6296      /* -----
6297         Bracket Expression
6298         -----
6299      */
6300      bracket_expression : '[' matching_list ']'
6301                    | '[' nonmatching_list ']'
6302                    ;
6303      matching_list    : bracket_list
6304                    ;
6305      nonmatching_list : '^' bracket_list
6306                    ;
6307      bracket_list     : follow_list
6308                    | follow_list '-'
6309                    ;
6310      follow_list      :      expression_term
6311                    | follow_list expression_term
6312                    ;
6313      expression_term : single_expression
6314                    | range_expression
6315                    ;

```

```

6316     single_expression : end_range
6317                          | character_class
6318                          | equivalence_class
6319                          ;
6320     range_expression : start_range end_range
6321                       | start_range '-'
6322                       ;
6323     start_range      : end_range '-'
6324                       ;
6325     end_range        : COLL_ELEM_SINGLE
6326                       | collating_symbol
6327                       ;
6328     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6329                       | Open_dot COLL_ELEM_MULTI Dot_close
6330                       | Open_dot META_CHAR Dot_close
6331                       ;
6332     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6333                       | Open_equal COLL_ELEM_MULTI Equal_close
6334                       ;
6335     character_class  : Open_colon class_name Colon_close
6336                       ;

```

6337 The BRE grammar does not permit **L\_ANCHOR** or **R\_ANCHOR** inside "`\(`" and "`\)`" (which  
6338 implies that '`^`' and '`$`' are ordinary characters). This reflects the semantic limits on the  
6339 application, as noted in [Section 9.3.8](#) (on page 187). Implementations are permitted to extend the  
6340 language to interpret '`^`' and '`$`' as anchors in these locations, and as such, conforming  
6341 applications cannot use unescaped '`^`' and '`$`' in positions inside "`\(`" and "`\)`" that might  
6342 be interpreted as anchors.

### 6343 9.5.3 ERE Grammar

6344 This section presents the grammar for extended regular expressions, excluding the bracket  
6345 expression grammar.

6346 **Note:** The bracket expression grammar and the associated `%token` lines are identical between BREs  
6347 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6348 %token ORD_CHAR QUOTED_CHAR DUP_COUNT
6349 %start extended_reg_exp
6350 %%
6351 /* -----
6352    Extended Regular Expression
6353    ----- */
6354 extended_reg_exp      : ERE_branch
6355                       | extended_reg_exp '|' ERE_branch
6356                       ;
6357 ERE_branch            : ERE_expression
6358                       | ERE_branch ERE_expression
6359                       ;
6360 ERE_expression        : one_char_or_coll_elem_ERE
6361                       | '^'
6362                       | '$'

```

```

6364         | '(' extended_reg_exp ')'
6365         | ERE_expression ERE_dupl_symbol
6366         ;
6367 one_char_or_coll_elem_ERE : ORD_CHAR
6368         | QUOTED_CHAR
6369         | '.'
6370         | bracket_expression
6371         ;
6372 ERE_dupl_symbol : '*'
6373         | '+'
6374         | '?'
6375         | '{' DUP_COUNT '}'
6376         | '{' DUP_COUNT ',' '}'
6377         | '{' DUP_COUNT ',' DUP_COUNT '}'
6378         ;

```

6379 The ERE grammar does not permit several constructs that previous sections specify as having  
6380 undefined results:

- 6381 • **ORD\_CHAR** preceded by a <backslash> character
- 6382 • One or more *ERE\_dupl\_symbols* appearing first in an ERE, or immediately following '|',  
6383 '^^', or '('
- 6384 • '{' not part of a valid *ERE\_dupl\_symbol*
- 6385 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or  
6386 immediately preceding ')'

6387 Implementations are permitted to extend the language to allow these. Conforming applications  
6388 cannot use such constructs.

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Directory Structure and Devices

### 10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

**/** The root directory.

**/dev** Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

**/tmp** A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

**/dev/null** An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

**/dev/tty** In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

**/dev/console** The **/dev/console** file is a generic name given to the system console (see [Section 3.384](#), on page 95). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of [Chapter 11](#) (on page 199).

6415 **10.2 Output Devices and Terminal Types**

6416 The utilities in the Shell and Utilities volume of POSIX.1-2008 historically have been  
 6417 implemented on a wide range of terminal types, but a conforming implementation need not  
 6418 support all features of all utilities on every conceivable terminal. POSIX.1-2008 states which  
 6419 features are optional for certain classes of terminals in the individual utility description sections.  
 6420 The implementation shall document in the system documentation which terminal types it  
 6421 supports and which of these features and utilities are not supported by each terminal.

6422 When a feature or utility is not supported on a specific terminal type, as allowed by  
 6423 POSIX.1-2008, and the implementation considers such a condition to be an error preventing use  
 6424 of the feature or utility, the implementation shall indicate such conditions through diagnostic  
 6425 messages or exit status values or both (as appropriate to the specific utility description) that  
 6426 inform the user that the terminal type lacks the appropriate capability.

6427 POSIX.1-2008 uses a notational convention based on historical practice that identifies some of  
 6428 the control characters defined in Section 7.3.1 (on page 139) in a manner easily remembered by  
 6429 users on many terminals. The correspondence between this “<control>-char” notation and the  
 6430 actual control characters is shown in the following table. When POSIX.1-2008 refers to a  
 6431 character by its <control>-name, it is referring to the actual control character shown in the Value  
 6432 column of the table, which is not necessarily the exact control key sequence on all terminals.  
 6433 Some terminals have keyboards that do not allow the direct transmission of all the non-  
 6434 alphanumeric characters shown. In such cases, the system documentation shall describe which  
 6435 data sequences transmitted by the terminal are interpreted by the system as representing the  
 6436 special characters.

6437 **Table 10-1** Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y	<EM>	<EM>
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\ <control>-]	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-^	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-_	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-?	<US>	<US>
<control>-P	<DLE>	<DLE>		<DEL>	<DEL>

6455 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that  
 6456 the keystrokes represent control-shift-letter sequences.

6457

Chapter 11

6458

## General Terminal Interface

6459

6460

6461

6462

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

### 11.1 Interface Characteristics

#### 11.1.1 Opening a Terminal Device File

6465

6466

6467

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

6468

Cases where applications do open a terminal device are as follows:

6469

6470

6471

6472

6473

6474

6475

6476

6477

6478

1. Opening `/dev/tty`, or the pathname returned by `ctermid()`, in order to obtain a file descriptor for the controlling terminal; see [Section 11.1.3](#) (on page 200).
2. Opening the slave side of a pseudo-terminal; see XSH `ptsname()`.
3. Opening a modem or similar piece of equipment connected by a serial line. In this case, the terminal parameters (see [Section 11.2](#), on page 205) may be initialized to default settings by the implementation in between the last close of the device by any process and the next open of the device, or they may persist from one use to the next. The terminal parameters can be set to values that ensure the terminal behaves in a conforming manner by means of the `O_TTY_INIT` open flag when opening a terminal device that is not already open in any process, or by executing the `stty` utility with the operand **sane**.

6479

6480

6481

6482

6483

As described in `open()`, opening a terminal device file with the `O_NONBLOCK` flag clear shall cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the terminal, or the `O_NONBLOCK` flag is specified in the `open()`, the `open()` function shall return a file descriptor without waiting for a connection to be established.

6484 **11.1.2 Process Groups**

6485 A terminal may have a foreground process group associated with it. This foreground process  
6486 group plays a special role in handling signal-generating input characters, as discussed in [Section](#)  
6487 [11.1.9](#) (on page 203).

6488 A command interpreter process supporting job control can allocate the terminal to different jobs,  
6489 or process groups, by placing related processes in a single process group and associating this  
6490 process group with the terminal. A terminal's foreground process group may be set or examined  
6491 by a process, assuming the permission requirements are met; see `tcgetpgrp()` and `tcsetpgrp()`.  
6492 The terminal interface aids in this allocation by restricting access to the terminal by processes  
6493 that are not in the current process group; see [Section 11.1.4](#) (on page 201).

6494 When there is no longer any process whose process ID or process group ID matches the  
6495 foreground process group ID, the terminal shall have no foreground process group. It is  
6496 unspecified whether the terminal has a foreground process group when there is a process whose  
6497 process ID matches the foreground process group ID, but whose process group ID does not. No  
6498 actions defined in POSIX.1-2008, other than allocation of a controlling terminal or a successful  
6499 call to `tcsetpgrp()`, shall cause a process group to become the foreground process group of the  
6500 terminal.

6501 **11.1.3 The Controlling Terminal**

6502 A terminal may belong to a process as its controlling terminal. Each process of a session that has  
6503 a controlling terminal has the same controlling terminal. A terminal may be the controlling  
6504 terminal for at most one session. The controlling terminal for a session is allocated by the session  
6505 leader in an implementation-defined manner. If a session leader has no controlling terminal, and  
6506 opens a terminal device file that is not already associated with a session without using the  
6507 `O_NOCTTY` option (see `open()`), it is implementation-defined whether the terminal becomes the  
6508 controlling terminal of the session leader. If a process which is not a session leader opens a  
6509 terminal file, or the `O_NOCTTY` option is used on `open()`, then that terminal shall not become  
6510 the controlling terminal of the calling process. When a controlling terminal becomes associated  
6511 with a session, its foreground process group shall be set to the process group of the session  
6512 leader.

6513 The controlling terminal is inherited by a child process during a `fork()` function call. A process  
6514 relinquishes its controlling terminal when it creates a new session with the `setsid()` function;  
6515 other processes remaining in the old session that had this terminal as their controlling terminal  
6516 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in  
6517 the current session) associated with the controlling terminal, it is unspecified whether all  
6518 processes that had that terminal as their controlling terminal cease to have any controlling  
6519 terminal. Whether and how a session leader can reacquire a controlling terminal after the  
6520 controlling terminal has been relinquished in this fashion is unspecified. A process does not  
6521 relinquish its controlling terminal simply by closing all of its file descriptors associated with the  
6522 controlling terminal if other processes continue to have it open.

6523 When a controlling process terminates, the controlling terminal is dissociated from the current  
6524 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by  
6525 other processes in the earlier session may be denied, with attempts to access the terminal treated  
6526 as if a modem disconnect had been sensed.

#### 6527 11.1.4 Terminal Access Control

6528 If a process is in the foreground process group of its controlling terminal, read operations shall  
 6529 be allowed, as described in Section 11.1.5. Any attempts by a process in a background process  
 6530 group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal  
 6531 unless one of the following special cases applies: if the reading process is ignoring or blocking  
 6532 the SIGTTIN signal, or if the process group of the reading process is orphaned, the *read()* shall  
 6533 return  $-1$ , with *errno* set to [EIO] and no signal shall be sent. The default action of the SIGTTIN  
 6534 signal shall be to stop the process to which it is sent. See `<signal.h>`.

6535 If a process is in the foreground process group of its controlling terminal, write operations shall  
 6536 be allowed as described in Section 11.1.8 (on page 203). Attempts by a process in a background  
 6537 process group to write to its controlling terminal shall cause the process group to be sent a  
 6538 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if  
 6539 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is  
 6540 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the  
 6541 process group of the writing process is orphaned, and the writing process is not ignoring or  
 6542 blocking the SIGTTOU signal, the *write()* shall return  $-1$ , with *errno* set to [EIO] and no signal  
 6543 shall be sent.

6544 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that  
 6545 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set  
 6546 (see Section 11.2.5 (on page 210), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, and  
 6547 *tcsetpgrp()*).

#### 6548 11.1.5 Input Processing and Reading Data

6549 A terminal device associated with a terminal device file may operate in full-duplex mode, so  
 6550 that data may arrive even while output is occurring. Each terminal device file has an input  
 6551 queue associated with it, into which incoming data is stored by the system before being read by  
 6552 a process. The system may impose a limit, {MAX\_INPUT}, on the number of bytes that may be  
 6553 stored in the input queue. The behavior of the system when this limit is exceeded is  
 6554 implementation-defined.

6555 Two general kinds of input processing are available, determined by whether the terminal device  
 6556 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 (on  
 6557 page 202) and Section 11.1.7 (on page 202). Additionally, input characters are processed  
 6558 according to the *c\_iflag* (see Section 11.2.2, on page 206) and *c\_lflag* (see Section 11.2.5, on page  
 6559 210) fields. Such processing can include “echoing”, which in general means transmitting input  
 6560 characters immediately back to the terminal when they are received from the terminal. This is  
 6561 useful for terminals that can operate in full-duplex mode.

6562 The manner in which data is provided to a process reading from a terminal device file is  
 6563 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether  
 6564 or not the O\_NONBLOCK flag is set by *open()* or *fcntl()*.

6565 If the O\_NONBLOCK flag is clear, then the read request shall be blocked until data is available  
 6566 or a signal has been received. If the O\_NONBLOCK flag is set, then the read request shall be  
 6567 completed, without blocking, in one of three ways:

- 6568 1. If there is enough data available to satisfy the entire request, the *read()* shall complete  
 6569 successfully and shall return the number of bytes read.
- 6570 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete  
 6571 successfully, having read as much data as possible, and shall return the number of bytes it  
 6572 was able to read.

6573 3. If there is no data available, the `read()` shall return `-1`, with `errno` set to `[EAGAIN]`.

6574 When data is available depends on whether the input processing mode is canonical or non-  
6575 canonical. Section 11.1.6 and Section 11.1.7 describe each of these input processing modes.

### 6576 11.1.6 Canonical Mode Input Processing

6577 In canonical mode input processing, terminal input is processed in units of lines. A line is  
6578 delimited by a <newline> character (NL), an end-of-file character (EOF), or an end-of-line (EOL)  
6579 character. See Section 11.1.9 (on page 203) for more information on EOF and EOL. This means  
6580 that a read request shall not return until an entire line has been typed or a signal has been  
6581 received. Also, no matter how many bytes are requested in the `read()` call, at most one line shall  
6582 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even  
6583 one, may be requested in a `read()` without losing information.

6584 If `{MAX_CANON}` is defined for this terminal device, it shall be a limit on the number of bytes  
6585 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If  
6586 `{MAX_CANON}` is not defined, there shall be no such limit; see `pathconf()`.

6587 Erase and kill processing occur when either of two special characters, the ERASE and KILL  
6588 characters (see Section 11.1.9, on page 203), is received. This processing shall affect data in the  
6589 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited  
6590 data makes up the current line. The ERASE character shall delete the last character in the current  
6591 line, if there is one. The KILL character shall delete all data in the current line, if there is any.  
6592 The ERASE and KILL characters shall have no effect if there is no data in the current line. The  
6593 ERASE and KILL characters themselves shall not be placed in the input queue.

### 6594 11.1.7 Non-Canonical Mode Input Processing

6595 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and  
6596 kill processing shall not occur. The values of the MIN and TIME members of the `c_cc` array are  
6597 used to determine how to process the bytes received. POSIX.1-2008 does not specify whether  
6598 the setting of `O_NONBLOCK` takes precedence over MIN or TIME settings. Therefore, if  
6599 `O_NONBLOCK` is set, `read()` may return immediately, regardless of the setting of MIN or TIME.  
6600 Also, if no data is available, `read()` may either return 0, or return `-1` with `errno` set to `[EAGAIN]`.

6601 MIN represents the minimum number of bytes that should be received when the `read()` function  
6602 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and  
6603 short-term data transmissions. If MIN is greater than `{MAX_INPUT}`, the response to the request  
6604 is undefined. The four possible values for MIN and TIME and their interactions are described  
6605 below.

#### 6606 Case A: MIN>0, TIME>0

6607 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is  
6608 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction  
6609 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall  
6610 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer  
6611 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN  
6612 bytes are received, the characters received to that point shall be returned to the user. Note that if  
6613 TIME expires at least one byte shall be returned because the timer would not have been enabled  
6614 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and  
6615 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is  
6616 in the buffer at the time of the `read()`, the result shall be as if data has been received immediately

6617 after the *read()*.

6618 **Case B: MIN>0, TIME=0**

6619 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A  
6620 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall  
6621 block until MIN bytes are received), or a signal is received. A program that uses case B to read  
6622 record-based terminal I/O may block indefinitely in the read operation.

6623 **Case C: MIN=0, TIME>0**

6624 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read  
6625 timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied  
6626 as soon as a single byte is received or the read timer expires. Note that in case C if the timer  
6627 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be  
6628 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely  
6629 waiting for a byte; if no byte is received within TIME\*0.1 seconds after the read is initiated, the  
6630 *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the  
6631 *read()*, the timer shall be started as if data has been received immediately after the *read()*.

6632 **Case D: MIN=0, TIME=0**

6633 The minimum of either the number of bytes requested or the number of bytes currently  
6634 available shall be returned without waiting for more bytes to be input. If no characters are  
6635 available, *read()* shall return a value of zero, having read no data.

6636 **11.1.8 Writing Data and Output Processing**

6637 When a process writes one or more bytes to a terminal device file, they are processed according  
6638 to the *c\_flag* field (see Section 11.2.3, on page 207). The implementation may provide a  
6639 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have  
6640 been scheduled for transmission to the device, but the transmission has not necessarily  
6641 completed. See *write()* for the effects of O\_NONBLOCK on *write()*.

6642 **11.1.9 Special Characters**

6643 Certain characters have special functions on input or output or both. These functions are  
6644 summarized as follows:

6645 **INTR** Special character on input, which is recognized if the ISIG flag is set. Generates a  
6646 SIGINT signal which is sent to all processes in the foreground process group for which  
6647 the terminal is the controlling terminal. If ISIG is set, the INTR character shall be  
6648 discarded when processed.

6649 **QUIT** Special character on input, which is recognized if the ISIG flag is set. Generates a  
6650 SIGQUIT signal which is sent to all processes in the foreground process group for  
6651 which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be  
6652 discarded when processed.

6653 **ERASE** Special character on input, which is recognized if the ICANON flag is set. Erases the  
6654 last character in the current line; see Section 11.1.6 (on page 202). It shall not erase  
6655 beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is  
6656 set, the ERASE character shall be discarded when processed.

6657	KILL	Special character on input, which is recognized if the ICANON flag is set. Deletes the entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL character shall be discarded when processed.
6658		
6659		
6660	EOF	Special character on input, which is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process without waiting for a <newline>, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be returned from the <i>read()</i> , representing an end-of-file indication. If ICANON is set, the EOF character shall be discarded when processed.
6661		
6662		
6663		
6664		
6665		
6666	NL	Special character on input, which is recognized if the ICANON flag is set. It is the line delimiter <newline>. It cannot be changed.
6667		
6668	EOL	Special character on input, which is recognized if the ICANON flag is set. It is an additional line delimiter, like NL.
6669		
6670	SUSP	If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character shall be discarded when processed.
6671		
6672		
6673	STOP	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character shall be discarded when processed.
6674		
6675		
6676		
6677	START	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character shall be discarded when processed.
6678		
6679		
6680		
6681	CR	Special character on input, which is recognized if the ICANON flag is set; it is the carriage-return character. When ICANON and ICRNL are set and IGNCR is not set, this character shall be translated into an NL, and shall have the same effect as an NL character.
6682		
6683		
6684		
6685		The NL and CR characters cannot be changed. It is implementation-defined whether the START and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP shall be changeable to suit individual tastes. Special character functions associated with changeable special control characters can be disabled individually.
6686		
6687		
6688		
6689		If two or more special characters have the same value, the function performed when that character is received is undefined.
6690		
6691		A special character is recognized not only by its value, but also by its context; for example, an implementation may support multi-byte sequences that have a meaning different from the meaning of the bytes when considered individually. Implementations may also support additional single-byte functions. These implementation-defined multi-byte or single-byte functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in this section.
6692		
6693		
6694		
6695		
6696		
6697	XSI	If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding <backslash> character, in which case no special function shall occur.
6698		

6699 **11.1.10 Modem Disconnect**

6700 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if  
 6701 CLOCAL is not set in the *c\_cflag* field for the terminal (see Section 11.2.4, on page 209), the  
 6702 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling  
 6703 terminal. Unless other arrangements have been made, this shall cause the controlling process to  
 6704 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of  
 6705 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-  
 6706 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*  
 6707 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent  
 6708 *write()* to the terminal device shall return  $-1$ , with *errno* set to [EIO], until the device is closed.

6709 **11.1.11 Closing a Terminal Device File**

6710 The last process to close a terminal device file shall cause any output to be sent to the device and  
 6711 any input to be discarded. If HUPCL is set in the control modes and the communications port  
 6712 supports a disconnect function, the terminal device shall perform a disconnect.

6713 **11.2 Parameters that Can be Set**6714 **11.2.1 The termios Structure**

6715 Routines that need to control certain terminal I/O characteristics shall do so by using the  
 6716 **termios** structure as defined in the `<termios.h>` header.

6717 Since the **termios** structure may include additional members, and the standard members may  
 6718 include both standard and non-standard modes, the structure should never be initialized  
 6719 directly by the application as this may cause the terminal to behave in a non-conforming  
 6720 manner. When opening a terminal device (other than a pseudo-terminal) that is not already open  
 6721 in any process, it should be opened with the `O_TTY_INIT` flag before initializing the structure  
 6722 using *tcgetattr()* to ensure that any non-standard elements of the **termios** structure are set to  
 6723 values that result in conforming behavior of the terminal interface.

6724 The members of the **termios** structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
<code>tcflag_t</code>		<i>c_iflag</i>	Input modes.
<code>tcflag_t</code>		<i>c_oflag</i>	Output modes.
<code>tcflag_t</code>		<i>c_cflag</i>	Control modes.
<code>tcflag_t</code>		<i>c_lflag</i>	Local modes.
<code>cc_t</code>	NCCS	<i>c_cc[ ]</i>	Control characters.

6732 The `tcflag_t` and `cc_t` types are defined in the `<termios.h>` header. They shall be unsigned  
 6733 integer types.

6734 **11.2.2 Input Modes**

6735 Values of the *c\_flag* field describe the basic terminal input control, and are composed of the  
 6736 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
 6737 symbols in this table are defined in [<termios.h>](#):

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

6751 In the context of asynchronous serial data transmission, a break condition shall be defined as a  
 6752 sequence of zero-valued bits that continues for more than the time to send one byte. The entire  
 6753 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a  
 6754 time equivalent to more than one byte. In contexts other than asynchronous serial data  
 6755 transmission, the definition of a break condition is implementation-defined.

6756 If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the  
 6757 input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the  
 6758 break condition shall flush the input and output queues, and if the terminal is the controlling  
 6759 terminal of a foreground process group, the break condition shall generate a single SIGINT  
 6760 signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break  
 6761 condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

6762 If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

6763 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than  
 6764 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is  
 6765 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid  
 6766 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff  
 6767 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be  
 6768 given to the application as a single byte 0x00.

6769 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking  
 6770 shall be disabled, allowing output parity generation without input parity errors. Note that  
 6771 whether input parity checking is enabled or disabled is independent of whether parity detection  
 6772 is enabled or disabled (see [Section 11.2.4](#), on page 209). If parity detection is enabled but input  
 6773 parity checking is disabled, the hardware to which the terminal is connected shall recognize the  
 6774 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6775 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits  
 6776 shall be processed.

6777 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a  
 6778 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a  
 6779 received CR character shall be translated into an NL character.

6780 If IXANY is set, any input character shall restart output that has been suspended.

6781 If IXON is set, start/stop output control shall be enabled. A received STOP character shall  
 6782 suspend output and a received START character shall restart output. When IXON is set, START  
 6783 and STOP characters are not read, but merely perform flow control functions. When IXON is not  
 6784 set, the START and STOP characters shall be read.

6785 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP  
 6786 characters, which are intended to cause the terminal device to stop transmitting data, as needed  
 6787 to prevent the input queue from overflowing and causing implementation-defined behavior,  
 6788 and shall transmit START characters, which are intended to cause the terminal device to resume  
 6789 transmitting data, as soon as the device can continue transmitting data without risk of  
 6790 overflowing the input queue. The precise conditions under which STOP and START characters  
 6791 are transmitted are implementation-defined.

6792 The initial input control value after *open()* is implementation-defined.

### 6793 11.2.3 Output Modes

6794 The *c\_oflag* field specifies the terminal interface's treatment of output, and is composed of the  
 6795 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
 6796 symbols in the following table are defined in `<termios.h>`:

Mask Name	Description
OPOST	Perform output processing.
XSI ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

6827 If OPOST is set, output data shall be post-processed as described below, so that lines of text are  
 6828 modified to appear appropriately on the terminal device; otherwise, characters shall be  
 6829 transmitted without change.

6830 XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is  
 6831 set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character  
 6832 shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is  
 6833 assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays  
 6834 specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed  
 6835 function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the  
 6836 CR character is actually transmitted.

6837 The delay bits specify how long transmission stops to allow for mechanical or other movement  
 6838 when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If  
 6839 OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful  
 6840 for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character  
 6841 shall be DEL; otherwise, NUL.

6842 If a <form-feed> or <vertical-tab> delay is specified, it shall last for about 2 seconds.

6843 Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be  
 6844 used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

6845 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be

6846 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall  
 6847 transmit two fill characters, and type 2 four fill characters.

6848 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be  
 6849 about 0.10 seconds. Type 3 specifies that <tab> characters shall be expanded into <space>  
 6850 characters. If OFILL is set, two fill characters shall be transmitted for any delay.

6851 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be  
 6852 transmitted.

6853 The actual delays depend on line speed and system load.

6854 The initial output control value after *open()* is implementation-defined.

#### 6855 11.2.4 Control Modes

6856 The *c\_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-  
 6857 inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in  
 6858 this table are defined in <*termios.h*>; not all values specified are required to be supported by the  
 6859 underlying hardware (if any). If the terminal is a pseudo-terminal, it is unspecified whether non-  
 6860 default values are unsupported, or are supported and emulated in software, or are handled by  
 6861 *tcsetattr()*, *tcgetattr()*, and the *stty* utility as if they are supported but have no effect on the  
 6862 behavior of the terminal interface.

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

6875 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in  
 6876 the following table are defined in <*termios.h*>. Not all values specified are required to be  
 6877 supported by the underlying hardware (if any). For pseudo-terminals, the input and output  
 6878 baud rates set in the **termios** structure need not affect the speed of data transmission through the  
 6879 terminal interface.

6880 **Note.** The term “baud” is used historically here, but is not technically correct. This is properly “bits  
 6881 per second”, which may not be the same as baud. However, the term is used because of the  
 6882 historical usage and understanding.

6883  
6884  
6885  
6886  
6887  
6888  
6889  
6890  
6891

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

6892 The following functions are provided for getting and setting the values of the input and output  
6893 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.  
6894 The effects on the terminal device shall not become effective and not all errors need be detected  
6895 until the *tcsetattr()* function is successfully called.

6896 The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not  
6897 set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-  
6898 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read.  
6899 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used;  
6900 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

6901 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

6902 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to  
6903 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity  
6904 shall be used.

6905 If HUPCL is set, the modem control lines for the port shall be lowered when the last process  
6906 with the port open closes the port or the process terminates. The modem connection shall be  
6907 broken.

6908 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If  
6909 CLOCAL is clear, the modem status lines shall be monitored.

6910 Under normal circumstances, a call to the *open()* function shall wait for the modem connection  
6911 to complete. However, if the O\_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set,  
6912 the *open()* function shall return immediately without waiting for the connection.

6913 If the object for which the control modes are set is not an asynchronous serial connection, some  
6914 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a  
6915 network connection to a terminal on another host, the baud rate need not be set on the  
6916 connection between that terminal and the machine to which it is directly connected.

6917 The initial hardware control value after *open()* is implementation-defined.

6918 **11.2.5 Local Modes**

6919 The *c\_lflag* field of the argument structure is used to control various functions. It is composed of  
6920 the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
6921 symbols in this table are defined in [<termios.h>](#).

6922  
6923  
6924  
6925  
6926  
6927  
6928  
6929  
6930  
6931

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

6932 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input  
6933 characters shall not be echoed.

6934 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if  
6935 possible, the last character in the current line from the display. If there is no character to erase, an  
6936 implementation may echo an indication that this was the case, or do nothing.

6937 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the  
6938 line from the display or shall echo the <newline> character after the KILL character.

6939 If ECHONL and ICANON are set, the <newline> character shall be echoed even if ECHO is not  
6940 set.

6941 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit  
6942 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as  
6943 described in [Section 11.1.6](#) (on page 202).

6944 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall  
6945 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired  
6946 between bytes. The time value represents tenths of a second. See [Section 11.1.7](#) (on page 202) for  
6947 more details.

6948 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is  
6949 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.  
6950 If IEXTEN is not set, implementation-defined functions shall not be recognized and the  
6951 corresponding input characters are processed as described for ICANON, ISIG, IXON, and  
6952 IXOFF.

6953 If ISIG is set, each input character shall be checked against the special control characters INTR,  
6954 QUIT, and SUSP. If an input character matches one of these control characters, the function  
6955 associated with that character shall be performed. If ISIG is not set, no checking shall be done.  
6956 Thus these special input functions are possible only if ISIG is set.

6957 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,  
6958 QUIT, and SUSP characters shall not be done.

6959 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to  
6960 write to its controlling terminal if it is not in the foreground process group for that terminal. This  
6961 signal, by default, stops the members of the process group. Otherwise, the output generated by  
6962 that process shall be output to the current output stream. Processes that are blocking or ignoring  
6963 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall  
6964 not be sent.

6965 The initial local control value after *open()* is implementation-defined.

6966 **11.2.6 Special Control Characters**

6967 The special control character values shall be defined by the array *c\_cc*. The subscript name and  
 6968 description for each element in both canonical and non-canonical modes are as follows:

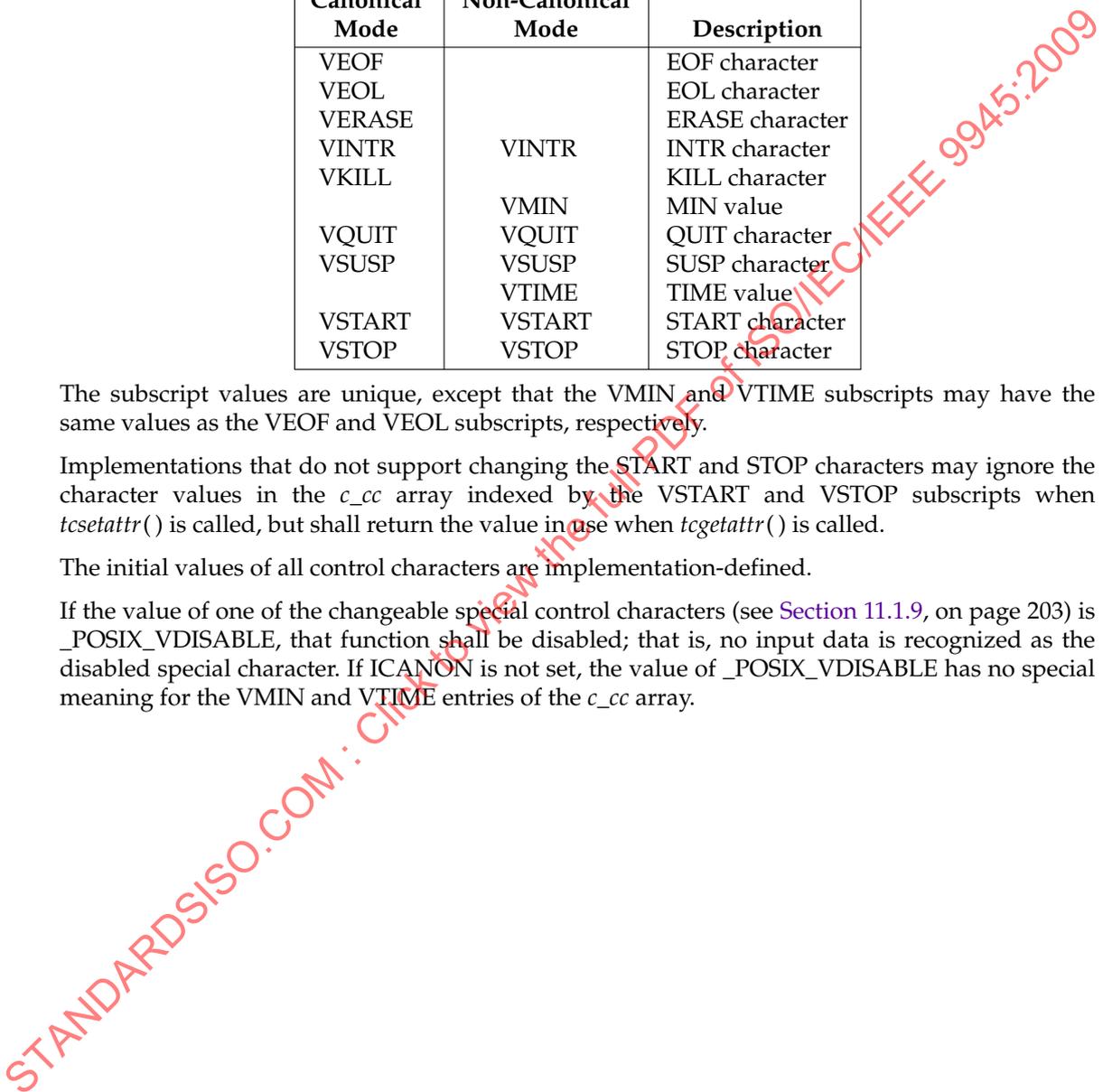
Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

6983 The subscript values are unique, except that the VMIN and VTIME subscripts may have the  
 6984 same values as the VEOF and VEOL subscripts, respectively.

6985 Implementations that do not support changing the START and STOP characters may ignore the  
 6986 character values in the *c\_cc* array indexed by the VSTART and VSTOP subscripts when  
 6987 *tcsetattr()* is called, but shall return the value in use when *tcgetattr()* is called.

6988 The initial values of all control characters are implementation-defined.

6989 If the value of one of the changeable special control characters (see Section 11.1.9, on page 203) is  
 6990 `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the  
 6991 disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special  
 6992 meaning for the VMIN and VTIME entries of the *c\_cc* array.



## 12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout POSIX.1-2008 for describing the arguments processed by the utilities.

Within POSIX.1-2008, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see XCU Section 2.9.1, on page 2316):

```
utility_name [-a] [-b] [-c option_argument]
             [-d|-e] [-f [option_argument]] [operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility\_name*. It is followed by options, option-arguments, and operands. The arguments that consist of <hyphen> characters and single letters or digits, such as 'a', are known as "options" (or, historically, "flags"). Certain options are followed by an "option-argument", as shown with [-c *option\_argument*]. The arguments following the last options and option-arguments are named "operands".
2. Option-arguments are shown separated from their options by <blank> characters, except when the option-argument is enclosed in the '[' and ']' notation to indicate that it is optional. This reflects the situation in which an optional option-argument (if present) is included within the same argument string as the option; for a mandatory option-argument, it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 215) require that the option be a separate argument from its option-argument and that option-arguments not be optional, but there are some exceptions in POSIX.1-2008 to ensure continued operation of historical applications:
  - a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-argument (as with [-c *option\_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument. However, a conforming implementation shall also permit applications to specify the option and option-argument in the same argument string without intervening <blank> characters.
  - b. If the SYNOPSIS shows an optional option-argument (as with [-f[*option\_argument*]] in the example), a conforming application shall place any option-argument for that option directly adjacent to the option in the same argument string, without intervening <blank> characters. If the utility receives an argument containing only the option, it shall behave as specified in its description for an omitted option-argument; it shall not treat the next argument (if any) as the option-argument for that option.

- 7031 3. Options are usually listed in alphabetical order unless this would make the utility  
 7032 description more confusing. There are no implied relationships between the options  
 7033 based upon the order in which they appear, unless otherwise stated in the OPTIONS  
 7034 section, or unless the exception in Guideline 11 of Section 12.2 (on page 215) applies. If an  
 7035 option that does not have option-arguments is repeated, the results are undefined, unless  
 7036 otherwise stated.
- 7037 4. Frequently, names of parameters that require substitution by actual values are shown  
 7038 with embedded <underscore> characters. Alternatively, parameters are shown as follows:  
 7039  
 <parameter name>
- 7040 The angle brackets are used for the symbolic grouping of a phrase representing a single  
 7041 parameter and conforming applications shall not include them in data submitted to the  
 7042 utility.
- 7043 5. When a utility has only a few permissible options, they are sometimes shown  
 7044 individually, as in the example. Utilities with many flags generally show all of the  
 7045 individual flags (that do not take option-arguments) grouped, as in:  
 7046  
 utility\_name [-abcDxyz] [-p arg] [operand]
- 7047 Utilities with very complex arguments may be shown as follows:  
 7048  
 utility\_name [options] [operands]
- 7049 6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a  
 7050 numeric value:
- 7051 • The number is interpreted as a decimal integer.
  - 7052 • Numerals in the range 0 to 2147483647 are syntactically recognized as numeric  
 7053 values.
  - 7054 • When the utility description states that it accepts negative numbers as operands or  
 7055 option-arguments, numerals in the range -2147483647 to 2147483647 are  
 7056 syntactically recognized as numeric values.
  - 7057 • Ranges greater than those listed here are allowed.
- 7058 This does not mean that all numbers within the allowable range are necessarily  
 7059 semantically correct. A standard utility that accepts an option-argument or operand that  
 7060 is to be interpreted as a number, and for which a range of values smaller than that shown  
 7061 above is permitted by the POSIX.1-2008, describes that smaller range along with the  
 7062 description of the option-argument or operand. If an error is generated, the utility's  
 7063 diagnostic message shall indicate that the value is out of the supported range, not that it  
 7064 is syntactically incorrect.
- 7065 7. Arguments or option-arguments enclosed in the ' [ ' and ' ] ' notation are optional and  
 7066 can be omitted. Conforming applications shall not include the ' [ ' and ' ] ' symbols in  
 7067 data submitted to the utility.
- 7068 8. Arguments separated by the ' | ' (<vertical-line>) bar notation are mutually-exclusive.  
 7069 Conforming applications shall not include the ' | ' symbol in data submitted to the utility.  
 7070 Alternatively, mutually-exclusive options and operands may be listed with multiple  
 7071 synopsis lines.

7072 For example:

```
7073 utility_name -d[-a] [-c option_argument] [operand...]
7074 utility_name [-a] [-b] [operand...]
```

7075 When multiple synopsis lines are given for a utility, it is an indication that the utility has  
 7076 mutually-exclusive arguments. These mutually-exclusive arguments alter the  
 7077 functionality of the utility so that only certain other arguments are valid in combination  
 7078 with one of the mutually-exclusive arguments. Only one of the mutually-exclusive  
 7079 arguments is allowed for invocation of the utility. Unless otherwise stated in an  
 7080 accompanying OPTIONS section, the relationships between arguments depicted in the  
 7081 SYNOPSIS sections are mandatory requirements placed on conforming applications. The  
 7082 use of conflicting mutually-exclusive arguments produces undefined results, unless a  
 7083 utility description specifies otherwise. When an option is shown without the '[' and  
 7084 ']' brackets, it means that option is required for that version of the SYNOPSIS. However,  
 7085 it is not required to be the first argument, as shown in the example above, unless  
 7086 otherwise stated.

7087 9. Ellipses ("...") are used to denote that one or more occurrences of an operand are  
 7088 allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero  
 7089 or more options or operands can be specified. The form:

```
7090 utility_name [-g option_argument]... [operand...]
```

7091 indicates that multiple occurrences of the option and its option-argument preceding the  
 7092 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See  
 7093 also Guideline 11 in Section 12.2.)

7094 The form:

```
7095 utility_name -f option_argument [-f option_argument]... [operand...]
```

7096 indicates that the `-f` option is required to appear at least once and may appear multiple  
 7097 times.

7098 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities  
 7099 volume of POSIX.1-2008, the indented lines following the initial line are continuation  
 7100 lines. An actual use of the command would appear on a single logical line.

## 7101 12.2 Utility Syntax Guidelines

7102 The following guidelines are established for the naming of utilities and for the specification of  
 7103 options, option-arguments, and operands. The `getopt()` function in the System Interfaces  
 7104 volume of POSIX.1-2008 assists utilities in handling options and operands that conform to these  
 7105 guidelines.

7106 Operands and option-arguments can contain characters not specified in the portable character  
 7107 set.

7108 The guidelines are intended to provide guidance to the authors of future utilities, such as those  
 7109 written specific to a local system or that are components of a larger application. Some of the  
 7110 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections  
 7111 describe the deviations.



*Utility Conventions**Utility Syntax Guidelines*

7156 It is recommended that all future utilities and applications use these guidelines to enhance user  
7157 portability. The fact that some historical utilities could not be changed (to avoid breaking  
7158 existing applications) should not deter this future goal.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

7159

Chapter 13

7160

**Headers**

7161

This chapter describes the contents of headers.

7162

7163

7164

7165

7166

Headers contain function prototypes, the definition of symbolic constants, common structures, preprocessor macros, and defined types. Each function in the System Interfaces volume of POSIX.1-2008 specifies the headers that an application shall include in order to use that function. In most cases, only one header is required. These headers are present on an application development system; they need not be present on the target execution system.

7167

**Format of Entries**

7168

7169

The entries in this chapter are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS and DESCRIPTION.

7170

**NAME**

7171

This section gives the name or names of the entry and briefly states its purpose.

7172

**SYNOPSIS**

7173

This section summarizes the use of the entry being described.

7174

**DESCRIPTION**

7175

This section describes the functionality of the header.

7176

**APPLICATION USAGE**

7177

7178

7179

7180

This section is informative. This section gives warnings and advice to application developers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of POSIX.1-2008, the normative material is to be taken as correct.

7181

**RATIONALE**

7182

7183

7184

This section is informative. This section contains historical information concerning the contents of this volume of POSIX.1-2008 and why features were included or discarded by the standard developers.

7185

**FUTURE DIRECTIONS**

7186

7187

7188

This section is informative. This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

7189

**SEE ALSO**

7190

This section is informative. This section gives references to related information.

7191

**CHANGE HISTORY**

7192

7193

This section is informative. This section shows the derivation of the entry and any significant changes that have been made to it.

## 7194 NAME

7195 aio.h — asynchronous input and output

## 7196 SYNOPSIS

7197 #include &lt;aio.h&gt;

## 7198 DESCRIPTION

7199 The <aio.h> header shall define the **aio\_cb** structure, which shall include at least the following  
7200 members:

7201	int	aio_fildes	File descriptor.
7202	off_t	aio_offset	File offset.
7203	volatile void *	aio_buf	Location of buffer.
7204	size_t	aio_nbytes	Length of transfer.
7205	int	aio_reqprio	Request priority offset.
7206	struct sigevent	aio_sigevent	Signal number and value.
7207	int	aio_lio_opcode	Operation to be performed.

7208 The <aio.h> header shall define the **off\_t**, **pthread\_attr\_t**, **size\_t**, and **ssize\_t** types as described  
7209 in <sys/types.h>.7210 The <aio.h> header shall define the **struct timespec** structure as described in <time.h>.7211 The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which  
7212 are described in the <signal.h> header.

7213 The &lt;aio.h&gt; header shall define the following symbolic constants:

7214 AIO\_ALLDONE A return value indicating that none of the requested operations could be  
7215 canceled since they are already complete.7216 AIO\_CANCELED A return value indicating that all requested operations have been  
7217 canceled.

7218 AIO\_NOTCANCELED

7219 A return value indicating that some of the requested operations could not  
7220 be canceled since they are in progress.7221 LIO\_NOP A *lio\_listio()* element operation option indicating that no transfer is  
7222 requested.7223 LIO\_NOWAIT A *lio\_listio()* synchronization operation indicating that the calling thread  
7224 is to continue execution while the *lio\_listio()* operation is being  
7225 performed, and no notification is given when the operation is complete.7226 LIO\_READ A *lio\_listio()* element operation option requesting a read.7227 LIO\_WAIT A *lio\_listio()* synchronization operation indicating that the calling thread  
7228 is to suspend until the *lio\_listio()* operation is complete.7229 LIO\_WRITE A *lio\_listio()* element operation option requesting a write.7230 The following shall be declared as functions and may also be defined as macros. Function  
7231 prototypes shall be provided.

```

7232 int aio_cancel(int, struct aiocb *);
7233 int aio_error(const struct aiocb *);
7234 int aio_fsync(int, struct aiocb *);
7235 int aio_read(struct aiocb *);
7236 ssize_t aio_return(struct aiocb *);
7237 int aio_suspend(const struct aiocb *const [], int,
```

```

7238             const struct timespec *);
7239 int         aio_write(struct aiocb *);
7240 int         lio_listio(int, struct aiocb *restrict const [restrict], int,
7241             struct sigevent *restrict);

```

7242 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>, <signal.h>, and <time.h>.

#### 7244 APPLICATION USAGE

7245 None.

#### 7246 RATIONALE

7247 None.

#### 7248 FUTURE DIRECTIONS

7249 None.

#### 7250 SEE ALSO

7251 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

7252 XSH *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_suspend()*, *aio\_write()*,  
7253 *fsync()*, *lio\_listio()*, *lseek()*, *read()*, *write()*

#### 7254 CHANGE HISTORY

7255 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 7256 Issue 6

7257 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7258 The description of the constants is expanded.

7259 The **restrict** keyword is added to the prototype for *lio\_listio()*.

#### 7260 Issue 7

7261 The <aio.h> header is moved from the Asynchronous Input and Output option to the Base.

7262 This reference page is clarified with respect to macros and symbolic constants, and type and  
7263 structure declarations are added.

**<arpa/inet.h>**

Headers

7264 **NAME**

7265 arpa/inet.h — definitions for internet operations

7266 **SYNOPSIS**

7267 #include &lt;arpa/inet.h&gt;

7268 **DESCRIPTION**7269 The **<arpa/inet.h>** header shall define the **in\_port\_t** and **in\_addr\_t** types as described in  
7270 **<netinet/in.h>**.7271 The **<arpa/inet.h>** header shall define the **in\_addr** structure as described in **<netinet/in.h>**.7272 IP6 The **<arpa/inet.h>** header shall define the **INET\_ADDRSTRLEN** and **INET6\_ADDRSTRLEN**  
7273 macros as described in **<netinet/in.h>**.7274 The following shall be declared as functions, or defined as macros, or both. If functions are  
7275 declared, function prototypes shall be provided.7276 uint32\_t htonl(uint32\_t);  
7277 uint16\_t htons(uint16\_t);  
7278 uint32\_t ntohl(uint32\_t);  
7279 uint16\_t ntohs(uint16\_t);7280 The **<arpa/inet.h>** header shall define the **uint32\_t** and **uint16\_t** types as described in  
7281 **<inttypes.h>**.7282 The following shall be declared as functions and may also be defined as macros. Function  
7283 prototypes shall be provided.7284 in\_addr\_t inet\_addr(const char \*);  
7285 char \*inet\_ntoa(struct in\_addr);  
7286 const char \*inet\_ntop(int, const void \*restrict, char \*restrict,  
7287 socklen\_t);  
7288 int inet\_pton(int, const char \*restrict, void \*restrict);7289 Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>**  
7290 and **<inttypes.h>**.7291 **APPLICATION USAGE**

7292 None.

7293 **RATIONALE**

7294 None.

7295 **FUTURE DIRECTIONS**

7296 None.

7297 **SEE ALSO**7298 **<inttypes.h>**, **<netinet/in.h>**7299 XSH **htonl()**, **inet\_addr()**, **inet\_ntop()**7300 **CHANGE HISTORY**

7301 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7302 The **restrict** keyword is added to the prototypes for **inet\_ntop()** and **inet\_pton()**.7303 **Issue 7**

7304 SD5-XBD-ERN-6 is applied.

7305 **NAME**

7306           assert.h — verify program assertion

7307 **SYNOPSIS**

7308           #include &lt;assert.h&gt;

7309 **DESCRIPTION**7310 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
7311 conflict between the requirements described here and the ISO C standard is unintentional. This  
7312 volume of POSIX.1-2008 defers to the ISO C standard.7313           The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is  
7314 not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this  
7315 header, the *assert()* macro shall be defined simply as:

7316           #define assert(ignore) ((void) 0)

7317           Otherwise, the macro behaves as described in *assert()*.7318           The *assert()* macro shall be redefined according to the current state of NDEBUG each time  
7319 <assert.h> is included.7320           The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is  
7321 suppressed in order to access an actual function, the behavior is undefined.7322 **APPLICATION USAGE**

7323           None.

7324 **RATIONALE**

7325           None.

7326 **FUTURE DIRECTIONS**

7327           None.

7328 **SEE ALSO**7329           XSH *assert()*7330 **CHANGE HISTORY**

7331           First released in Issue 1. Derived from Issue 1 of the SVID.

7332 **Issue 6**7333           The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999  
7334 standard.

**<complex.h>**7335 **NAME**7336 `complex.h` — complex arithmetic7337 **SYNOPSIS**7338 `#include <complex.h>`7339 **DESCRIPTION**

7340 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 7341 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7342 volume of POSIX.1-2008 defers to the ISO C standard.

7343 The **<complex.h>** header shall define the following macros:7344 `complex` Expands to `_Complex`.7345 `_Complex_I` Expands to a constant expression of type `const float _Complex`, with the value  
7346 of the imaginary unit (that is, a number  $i$  such that  $i^2 = -1$ ).7347 `imaginary` Expands to `_Imaginary`.7348 `_Imaginary_I` Expands to a constant expression of type `const float _Imaginary` with the  
7349 value of the imaginary unit.7350 `I` Expands to either `_Imaginary_I` or `_Complex_I`. If `_Imaginary_I` is not defined,  
7351 `I` expands to `_Complex_I`.7352 The macros `imaginary` and `_Imaginary_I` shall be defined if and only if the implementation  
7353 supports imaginary types.7354 An application may undefine and then, perhaps, redefine the `complex`, `imaginary`, and `I` macros.7355 The following shall be declared as functions and may also be defined as macros. Function  
7356 prototypes shall be provided.

```

7357 double          cabs(double complex);
7358 float           cabsf(float complex);
7359 long double     cabsl(long double complex);
7360 double complex cacos(double complex);
7361 float complex  cacosf(float complex);
7362 double complex cacosh(double complex);
7363 float complex  cacoshf(float complex);
7364 long double complex cacoshl(long double complex);
7365 long double complex cacosl(long double complex);
7366 double         carg(double complex);
7367 float          cargf(float complex);
7368 long double    cargl(long double complex);
7369 double complex casin(double complex);
7370 float complex casinf(float complex);
7371 double complex casinh(double complex);
7372 float complex casinhf(float complex);
7373 long double complex casinhl(long double complex);
7374 long double complex casinl(long double complex);
7375 double complex catan(double complex);
7376 float complex  catanf(float complex);
7377 double complex catanh(double complex);
7378 float complex  catanhf(float complex);
7379 long double complex catanhl(long double complex);
7380 long double complex catanl(long double complex);

```

## Headers

## &lt;complex.h&gt;

7381	double complex	ccos(double complex);
7382	float complex	ccosf(float complex);
7383	double complex	ccosh(double complex);
7384	float complex	ccoshf(float complex);
7385	long double complex	ccoshl(long double complex);
7386	long double complex	ccosl(long double complex);
7387	double complex	cexp(double complex);
7388	float complex	cexpf(float complex);
7389	long double complex	cexpl(long double complex);
7390	double	cimag(double complex);
7391	float	cimagf(float complex);
7392	long double	cimagl(long double complex);
7393	double complex	clog(double complex);
7394	float complex	clogf(float complex);
7395	long double complex	clogl(long double complex);
7396	double complex	conj(double complex);
7397	float complex	conjf(float complex);
7398	long double complex	conjl(long double complex);
7399	double complex	cpow(double complex, double complex);
7400	float complex	cpowf(float complex, float complex);
7401	long double complex	cpowl(long double complex, long double complex);
7402	double complex	cproj(double complex);
7403	float complex	cprojf(float complex);
7404	long double complex	cprojl(long double complex);
7405	double	creal(double complex);
7406	float	crealf(float complex);
7407	long double	creall(long double complex);
7408	double complex	csin(double complex);
7409	float complex	csinf(float complex);
7410	double complex	csinh(double complex);
7411	float complex	csinhf(float complex);
7412	long double complex	csinhl(long double complex);
7413	long double complex	csinl(long double complex);
7414	double complex	csqrt(double complex);
7415	float complex	csqrtf(float complex);
7416	long double complex	csqrtl(long double complex);
7417	double complex	ctan(double complex);
7418	float complex	ctanf(float complex);
7419	double complex	ctanh(double complex);
7420	float complex	ctanhf(float complex);
7421	long double complex	ctanhl(long double complex);
7422	long double complex	ctanl(long double complex);

**<complex.h>**7423 **APPLICATION USAGE**

7424 Values are interpreted as radians, not degrees.

7425 **RATIONALE**7426 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the  
7427 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the  
7428 imaginary unit by following the inclusion of the **<complex.h>** header with:7429 

```
#undef I
7430 #define j _Imaginary_I
```

7431 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a  
7432 sufficiently convenient and more generally useful notation for imaginary terms. The  
7433 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or  
7434 notational convenience will not result in widening types.7435 On systems with imaginary types, the application has the ability to control whether use of the  
7436 macro *I* introduces an imaginary type, by explicitly defining *I* to be `_Imaginary_I` or `_Complex_I`.  
7437 Disallowing imaginary types is useful for some applications intended to run on  
7438 implementations without support for such types.7439 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.7440 The `cis()` function ( $\cos(x) + I*\sin(x)$ ) was considered but rejected because its implementation is  
7441 easy and straightforward, even though some implementations could compute sine and cosine  
7442 more efficiently in tandem.7443 **FUTURE DIRECTIONS**7444 The following function names and the same names suffixed with *f* or *l* are reserved for future  
7445 use, and may be added to the declarations in the **<complex.h>** header.7446 

<code>cerf()</code>	<code>cexpm1()</code>	<code>clog2()</code>
<code>cerfc()</code>	<code>clog10()</code>	<code>clgamma()</code>
<code>cexp2()</code>	<code>clog1p()</code>	<code>ctgamma()</code>

7449 **SEE ALSO**7450 XSH `cabs()`, `cacos()`, `cacosh()`, `carg()`, `casin()`, `casinh()`, `catan()`, `catanh()`, `ccos()`, `ccosh()`, `cexp()`,  
7451 `cimag()`, `clog()`, `conj()`, `cpow()`, `cproj()`, `creal()`, `csin()`, `csinh()`, `csqrt()`, `ctan()`, `ctanh()`7452 **CHANGE HISTORY**

7453 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

## Headers

## &lt;cpio.h&gt;

7454 **NAME**

7455 cpio.h — cpio archive values

7456 **SYNOPSIS**

7457 #include &lt;cpio.h&gt;

7458 **DESCRIPTION**7459 The <cpio.h> header shall define the symbolic constants needed by the *c\_mode* field of the *cpio*  
7460 archive format, with the names and values given in the following table:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7482 The &lt;cpio.h&gt; header shall define the following symbolic constant as a string:

7483 MAGIC "070707"

7484 **APPLICATION USAGE**

7485 None.

7486 **RATIONALE**

7487 None.

7488 **FUTURE DIRECTIONS**

7489 None.

7490 **SEE ALSO**7491 XCU *pax*7492 **CHANGE HISTORY**7493 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988  
7494 standard.7495 **Issue 6**7496 The SEE ALSO is updated to refer to *pax*.

**<pio.h>**

*Headers*

7497 **Issue 7**

7498 The **<pio.h>** header is moved from the XSI option to the Base.

7499 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;ctype.h&gt;

## 7500 NAME

7501           ctype.h — character types

## 7502 SYNOPSIS

7503           #include &lt;ctype.h&gt;

## 7504 DESCRIPTION

7505 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 7506       Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 7507       enable the visibility of these symbols in this header.

7508       The <ctype.h> header shall define the **locale\_t** type as described in <locale.h>, representing a  
 7509       locale object.

7510       The following shall be declared as functions and may also be defined as macros. Function  
 7511       prototypes shall be provided for use with ISO C standard compilers.

```

7512       int    isalnum(int);
7513 CX       int    isalnum_l(int, locale_t);
7514       int    isalpha(int);
7515 CX       int    isalpha_l(int, locale_t);
7516 OB XSI   int    isascii(int);
7517       int    isblank(int);
7518 CX       int    isblank_l(int, locale_t);
7519       int    iscntrl(int);
7520 CX       int    iscntrl_l(int, locale_t);
7521       int    isdigit(int);
7522 CX       int    isdigit_l(int, locale_t);
7523       int    isgraph(int);
7524 CX       int    isgraph_l(int, locale_t);
7525       int    islower(int);
7526 CX       int    islower_l(int, locale_t);
7527       int    isprint(int);
7528 CX       int    isprint_l(int, locale_t);
7529       int    ispunct(int);
7530 CX       int    ispunct_l(int, locale_t);
7531       int    isspace(int);
7532 CX       int    isspace_l(int, locale_t);
7533       int    isupper(int);
7534 CX       int    isupper_l(int, locale_t);
7535       int    isxdigit(int);
7536 CX       int    isxdigit_l(int, locale_t);
7537 OB XSI   int    toascii(int);
7538       int    tolower(int);
7539 CX       int    tolower_l(int, locale_t);
7540       int    toupper(int);
7541 CX       int    toupper_l(int, locale_t);
  
```

7542       The <ctype.h> header shall define the following as macros:

```

7543 OB XSI   int    __toupper(int);
7544       int    __tolower(int);
  
```

## &lt;ctype.h&gt;

Headers

7545 **APPLICATION USAGE**

7546 None.

7547 **RATIONALE**

7548 None.

7549 **FUTURE DIRECTIONS**

7550 None.

7551 **SEE ALSO**7552 [<locale.h>](#)

7553 XSH Section 2.2 (on page 468), [isalnum\(\)](#), [isalpha\(\)](#), [isascii\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#),  
 7554 [isgraph\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#), [isupper\(\)](#), [isxdigit\(\)](#), [mblen\(\)](#), [mbstowcs\(\)](#),  
 7555 [mbtowc\(\)](#), [setlocale\(\)](#), [toascii\(\)](#), [tolower\(\)](#), [\\_tolower\(\)](#), [toupper\(\)](#), [\\_toupper\(\)](#), [wcstombs\(\)](#), [wctomb\(\)](#)

7556 **CHANGE HISTORY**

7557 First released in Issue 1. Derived from Issue 1 of the SVID.

7558 **Issue 6**

7559 Extensions beyond the ISO C standard are marked.

7560 **Issue 7**

7561 SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for  
 7562 consistency.

7563 The \*\_l() functions are added from The Open Group Technical Standard, 2006, Extended API Set  
 7564 Part 4.

7565 **NAME**7566 `dirent.h` — format of directory entries7567 **SYNOPSIS**7568 `#include <dirent.h>`7569 **DESCRIPTION**

7570 The internal format of directories is unspecified.

7571 The <**dirent.h**> header shall define the following type:7572 **DIR** A type representing a directory stream. The **DIR** type may be an incomplete type:7573 It shall also define the structure **dirent** which shall include the following members:7574 XSI `ino_t d_ino` File serial number.7575 `char d_name[]` Name of entry.7576 XSI The <**dirent.h**> header shall define the `ino_t` type as described in <**sys/types.h**>.7577 The character array `d_name` is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed `{NAME_MAX}`.

7579 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```

7581 int      alphasort(const struct dirent **, const struct dirent **);
7582 int      closedir(DIR *);
7583 int      dirfd(DIR *);
7584 DIR      *fdopendir(int);
7585 DIR      *opendir(const char *);
7586 struct dirent *readdir(DIR *);
7587 int      readdir_r(DIR *restrict, struct dirent *restrict,
7588                struct dirent **restrict);
7589 void     rewinddir(DIR *);
7590 int      scandir(const char *, struct dirent ***,
7591                int (*)(const struct dirent *),
7592                int (*)(const struct dirent **),
7593                const struct dirent **);
7594 XSI     void seekdir(DIR *, long);
7595 long     telldir(DIR *);

```

7596 **APPLICATION USAGE**

7597 None.

7598 **RATIONALE**

7599 Information similar to that in the <**dirent.h**> header is contained in a file <**sys/dir.h**> in 4.2 BSD  
7600 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of  
7601 POSIX.1-2008 is **struct direct**. The filename was changed because the name <**sys/dir.h**> was also  
7602 used in earlier implementations to refer to definitions related to the older access method; this  
7603 produced name conflicts. The name of the structure was changed because this volume of  
7604 POSIX.1-2008 does not completely define what is in the structure, so it could be different on  
7605 some implementations from **struct direct**.

7606 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:7607 `sizeof(d_name)`

7608 is incorrect; use:

## &lt;dirent.h&gt;

Headers

7609            `strlen(d_name)`

7610            instead.

7611            The array of **char** *d\_name* is not a fixed size. Implementations may need to declare **struct dirent** with an array size for *d\_name* of 1, but the actual number of characters provided matches (or only slightly exceeds) the length of the filename.

7614    **FUTURE DIRECTIONS**

7615            None.

7616    **SEE ALSO**

7617            [<sys/types.h>](#)

7618            XSH *alphasort()*, *closedir()*, *dirfd()*, *fdopendir()*, *readdir()*, *rewinddir()*, *seekdir()*, *telldir()*

7619    **CHANGE HISTORY**

7620            First released in Issue 2.

7621    **Issue 5**

7622            The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7623    **Issue 6**

7624            The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.

7625            The **restrict** keyword is added to the prototype for *readdir\_r()*.

7626    **Issue 7**

7627            The *alphasort()*, *dirfd()*, and *scandir()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

7629            The *fopendir()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

7631            Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the **DIR** type.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;dlfcn.h&gt;

7633 **NAME**7634 `dlfcn.h` — dynamic linking7635 **SYNOPSIS**7636 `#include <dlfcn.h>`7637 **DESCRIPTION**7638 The `<dlfcn.h>` header shall define at least the following symbolic constants for use in the  
7639 construction of a `dlopen()` *mode* argument:7640 `RTLD_LAZY` Relocations are performed at an implementation-defined time.7641 `RTLD_NOW` Relocations are performed when the object is loaded.7642 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.7643 `RTLD_LOCAL` All symbols are not made available for relocation processing by other  
7644 modules.7645 The following shall be declared as functions and may also be defined as macros. Function  
7646 prototypes shall be provided.7647 `int dlclose(void *);`  
7648 `char *dlerror(void);`  
7649 `void *dlopen(const char *, int);`  
7650 `void *dlsym(void *restrict, const char *restrict);`7651 **APPLICATION USAGE**

7652 None.

7653 **RATIONALE**

7654 None.

7655 **FUTURE DIRECTIONS**

7656 None.

7657 **SEE ALSO**7658 XSH `dlclose()`, `dlerror()`, `dlopen()`, `dlsym()`7659 **CHANGE HISTORY**

7660 First released in Issue 5.

7661 **Issue 6**7662 The `restrict` keyword is added to the prototype for `dlsym()`.7663 **Issue 7**7664 The `<dlfcn.h>` header is moved from the XSI option to the Base.

7665 This reference page is clarified with respect to macros and symbolic constants.

**<errno.h>**7666 **NAME**7667 `errno.h` — system error numbers7668 **SYNOPSIS**7669 `#include <errno.h>`7670 **DESCRIPTION**7671 **CX** Some of the functionality described on this reference page extends the ISO C standard. Any  
7672 conflict between the requirements described here and the ISO C standard is unintentional. This  
7673 volume of POSIX.1-2008 defers to the ISO C standard.

7674 The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7675 The **<errno.h>** header shall provide a declaration or definition for *errno*. The symbol *errno* shall  
7676 expand to a modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an  
7677 identifier declared with external linkage. If a macro definition is suppressed in order to access an  
7678 actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.7679 The **<errno.h>** header shall define the following macros which shall expand to integer constant  
7680 expressions with type **int**, distinct positive values (except as noted below), and which shall be  
7681 suitable for use in **#if** preprocessing directives:

7682 [E2BIG] Argument list too long.

7683 [EACCES] Permission denied.

7684 [EADDRINUSE] Address in use.

7685 [EADDRNOTAVAIL] Address not available.

7686 [EAFNOSUPPORT] Address family not supported.

7687 [EAGAIN] Resource unavailable, try again (may be the same value as  
7688 [EWOULDBLOCK]).

7689 [EALREADY] Connection already in progress.

7690 [EBADF] Bad file descriptor.

7691 [EBADMSG] Bad message.

7692 [EBUSY] Device or resource busy.

7693 [ECANCELED] Operation canceled.

7694 [ECHILD] No child processes.

7695 [ECONNABORTED] Connection aborted.

7696 [ECONNREFUSED] Connection refused.

7697 [ECONNRESET] Connection reset.

7698 [EDEADLK] Resource deadlock would occur.

7699 [EDESTADDRREQ] Destination address required.

7700 [EDOM] Mathematics argument out of domain of function.

7701 [EDQUOT] Reserved.

7702 [EEXIST] File exists.

## Headers

## &lt;errno.h&gt;

7703	[EFAULT]	Bad address.
7704	[EFBIG]	File too large.
7705	[EHOSTUNREACH]	Host is unreachable.
7706	[EIDRM]	Identifier removed.
7707	[EILSEQ]	Illegal byte sequence.
7708	[EINPROGRESS]	Operation in progress.
7709	[EINTR]	Interrupted function.
7710	[EINVAL]	Invalid argument.
7711	[EIO]	I/O error.
7712	[EISCONN]	Socket is connected.
7713	[EISDIR]	Is a directory.
7714	[ELOOP]	Too many levels of symbolic links.
7715	[EMFILE]	File descriptor value too large.
7716	[EMLINK]	Too many links.
7717	[EMSGSIZE]	Message too large.
7718	[EMULTIHOP]	Reserved.
7719	[ENAMETOOLONG]	Filename too long.
7720	[ENETDOWN]	Network is down.
7721	[ENETRESET]	Connection aborted by network.
7722	[ENETUNREACH]	Network unreachable.
7723	[ENFILE]	Too many files open in system.
7724	[ENOBUFS]	No buffer space available.
7725	OB XSR [ENODATA]	No message is available on the STREAM head read queue.
7726	[ENODEV]	No such device.
7727	[ENOENT]	No such file or directory.
7728	[ENOEXEC]	Executable file format error.
7729	[ENOLCK]	No locks available.
7730	[ENOLINK]	Reserved.
7731	[ENOMEM]	Not enough space.
7732	[ENOMSG]	No message of the desired type.
7733	[ENOPROTOPT]	Protocol not available.
7734	[ENOSPC]	No space left on device.
7735	OB XSR [ENOSR]	No STREAM resources.

## &lt;errno.h&gt;

Headers

7736	OB XSR	[ENOSTR]	Not a STREAM.
7737		[ENOSYS]	Function not supported.
7738		[ENOTCONN]	The socket is not connected.
7739		[ENOTDIR]	Not a directory.
7740		[ENOTEMPTY]	Directory not empty.
7741		[ENOTRECOVERABLE]	
7742			State not recoverable.
7743		[ENOTSOCK]	Not a socket.
7744		[ENOTSUP]	Not supported (may be the same value as [EOPNOTSUPP]).
7745		[ENOTTY]	Inappropriate I/O control operation.
7746		[ENXIO]	No such device or address.
7747		[EOPNOTSUPP]	Operation not supported on socket (may be the same value as
7748			[ENOTSUP]).
7749		[EOVERFLOW]	Value too large to be stored in data type.
7750		[EOWNERDEAD]	Previous owner died.
7751		[EPERM]	Operation not permitted.
7752		[EPIPE]	Broken pipe.
7753		[EPROTO]	Protocol error.
7754		[EPROTONOSUPPORT]	
7755			Protocol not supported.
7756		[EPROTOTYPE]	Protocol wrong type for socket.
7757		[ERANGE]	Result too large.
7758		[EROFS]	Read-only file system.
7759		[ESPIPE]	Invalid seek.
7760		[ESRCH]	No such process.
7761		[ESTALE]	Reserved.
7762	OB XSR	[ETIME]	Stream <i>ioctl()</i> timeout.
7763		[ETIMEDOUT]	Connection timed out.
7764		[ETXTBSY]	Text file busy.
7765		[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7766		[EXDEV]	Cross-device link.

- 7767 **APPLICATION USAGE**  
 7768 Additional error numbers may be defined on conforming systems; see the System Interfaces  
 7769 volume of POSIX.1-2008.
- 7770 **RATIONALE**  
 7771 None.
- 7772 **FUTURE DIRECTIONS**  
 7773 None.
- 7774 **SEE ALSO**  
 7775 XSH Section 2.3 (on page 477)
- 7776 **CHANGE HISTORY**  
 7777 First released in Issue 1. Derived from Issue 1 of the SVID.
- 7778 **Issue 5**  
 7779 Updated for alignment with the POSIX Realtime Extension.
- 7780 **Issue 6**  
 7781 The following new requirements on POSIX implementations derive from alignment with the  
 7782 Single UNIX Specification:  
 7783 • The majority of the error conditions previously marked as extensions are now mandatory,  
 7784 except for the STREAMS-related error conditions.  
 7785 Values for *errno* are now required to be distinct positive values rather than non-zero values. This  
 7786 change is for alignment with the ISO/IEC 9899:1999 standard.
- 7787 **Issue 7**  
 7788 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and  
 7789 [EOPNOTSUPP] to be the same values.  
 7790 The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group  
 7791 Technical Standard, 2006, Extended API Set Part 2.  
 7792 Functionality relating to the XSI STREAMS option is marked obsolescent.  
 7793 Functionality relating to the Threads option is moved to the Base.  
 7794 This reference page is clarified with respect to macros and symbolic constants.

**<fcntl.h>**

Headers

7795 **NAME**7796 `fcntl.h` — file control options7797 **SYNOPSIS**7798 `#include <fcntl.h>`7799 **DESCRIPTION**7800 The **<fcntl.h>** header shall define the following symbolic constants for the *cmd* argument used  
7801 by *fcntl()*. The values shall be unique and shall be suitable for use in **#if** preprocessing  
7802 directives.7803 `F_DUPFD` Duplicate file descriptor.7804 `F_DUPFD_CLOEXEC`7805 Duplicate file descriptor with the close-on-exec flag `FD_CLOEXEC` set.7806 `F_GETFD` Get file descriptor flags.7807 `F_SETFD` Set file descriptor flags.7808 `F_GETFL` Get file status flags and file access modes.7809 `F_SETFL` Set file status flags.7810 `F_GETLK` Get record locking information.7811 `F_SETLK` Set record locking information.7812 `F_SETLKW` Set record locking information; wait if blocked.7813 `F_GETOWN` Get process or process group ID to receive SIGURG signals.7814 `F_SETOWN` Set process or process group ID to receive SIGURG signals.7815 The **<fcntl.h>** header shall define the following symbolic constant used for the *fcntl()* file  
7816 descriptor flags, which shall be suitable for use in **#if** preprocessing directives.7817 `FD_CLOEXEC` Close the file descriptor upon execution of an *exec* family function.7818 The **<fcntl.h>** header shall also define the following symbolic constants for the *l\_type* argument  
7819 used for record locking with *fcntl()*. The values shall be unique and shall be suitable for use in  
7820 **#if** preprocessing directives.7821 `F_RDLCK` Shared or read lock.7822 `F_UNLCK` Unlock.7823 `F_WRLCK` Exclusive or write lock.7824 The **<fcntl.h>** header shall define the values used for *l\_whence*, `SEEK_SET`, `SEEK_CUR`, and  
7825 `SEEK_END` as described in **<stdio.h>**.7826 The **<fcntl.h>** header shall define the following symbolic constants as file creation flags for use  
7827 in the *oflag* value to *open()* and *openat()*. The values shall be bitwise-distinct and shall be  
7828 suitable for use in **#if** preprocessing directives.7829 `O_CREAT` Create file if it does not exist.7830 `O_EXCL` Exclusive use flag.7831 `O_NOCTTY` Do not assign controlling terminal.7832 `O_TRUNC` Truncate flag.

## Headers

## &lt;fcntl.h&gt;

7833		O_TTY_INIT	Set the <b>termios</b> structure terminal parameters to a state that provides conforming behavior; see <a href="#">Section 11.2</a> (on page 205).
7834			
7835			The O_TTY_INIT flag can have the value zero and in this case it need not be bitwise-distinct from the other flags.
7836			
7837			The <fcntl.h> header shall define the following symbolic constants for use as file status flags for <i>open()</i> , <i>openat()</i> , and <i>fcntl()</i> . The values shall be suitable for use in <b>#if</b> preprocessing directives.
7838			
7839		O_APPEND	Set append mode.
7840	SIO	O_DSYNC	Write according to synchronized I/O data integrity completion.
7841		O_NONBLOCK	Non-blocking mode.
7842	SIO	O_RSYNC	Synchronized read I/O operations.
7843		O_SYNC	Write according to synchronized I/O file integrity completion.
7844			The <fcntl.h> header shall define the following symbolic constant for use as the mask for file access modes. The value shall be suitable for use in <b>#if</b> preprocessing directives.
7845			
7846		O_ACCMODE	Mask for file access modes.
7847			The <fcntl.h> header shall define the following symbolic constants for use as the file access modes for <i>open()</i> , <i>openat()</i> , and <i>fcntl()</i> . The values shall be suitable for use in <b>#if</b> preprocessing directives.
7848			
7849			
7850		O_EXEC	Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.
7851			
7852		O_RDONLY	Open for reading only.
7853		O_RDWR	Open for reading and writing.
7854		O_SEARCH	Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.
7855			
7856		O_WRONLY	Open for writing only.
7857			The <fcntl.h> header shall define the symbolic constants for file modes for use as values of <b>mode_t</b> as described in <sys/stat.h>.
7858			
7859			The <fcntl.h> header shall define the following symbolic constant as a special value used in place of a file descriptor for the <i>*at()</i> functions which take a directory file descriptor as a parameter:
7860			
7861			
7862		AT_FDCWD	Use the current working directory to determine the target of relative file paths.
7863			The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>faccessat()</i> :
7864			
7865		AT_EACCESS	Check access using effective user and group ID.
7866			The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>fstatat()</i> , <i>fchmodat()</i> , <i>fchownat()</i> , and <i>utimensat()</i> :
7867			
7868		AT_SYMLINK_NOFOLLOW	
7869			Do not follow symbolic links.
7870			The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by <i>linkat()</i> :
7871			

## &lt;fcntl.h&gt;

7872 AT\_SYMLINK\_FOLLOW  
7873 Follow symbolic link.

7874 The <fcntl.h> header shall define the following symbolic constants as values for the flag used by  
7875 *open()* and *openat()*:

7876 O\_CLOEXEC The FD\_CLOEXEC flag associated with the new descriptor shall be set to close  
7877 the file descriptor upon execution of an *exec* family function.

7878 O\_DIRECTORY Fail if not a directory.

7879 O\_NOFOLLOW Do not follow symbolic links.

7880 The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by  
7881 *unlinkat()*:

7882 AT\_REMOVEDIR  
7883 Remove directory instead of file.

7884 ADV The <fcntl.h> header shall define the following symbolic constants for the *advise* argument used  
7885 by *posix\_fadvise()*:

7886 POSIX\_FADV\_DONTNEED  
7887 The application expects that it will not access the specified data in the near future.

7888 POSIX\_FADV\_NOREUSE  
7889 The application expects to access the specified data once and then not reuse it thereafter.

7890 POSIX\_FADV\_NORMAL  
7891 The application has no advice to give on its behavior with respect to the specified data. It is  
7892 the default characteristic if no advice is given for an open file.

7893 POSIX\_FADV\_RANDOM  
7894 The application expects to access the specified data in a random order.

7895 POSIX\_FADV\_SEQUENTIAL  
7896 The application expects to access the specified data sequentially from lower offsets to higher  
7897 offsets.

7898 POSIX\_FADV\_WILLNEED  
7899 The application expects to access the specified data in the near future.

7900 The <fcntl.h> header shall define the **flock** structure describing a file lock. It shall include the  
7901 following members:

7902 short l\_type Type of lock; F\_RDLCK, F\_WRLCK, F\_UNLCK.  
7903 short l\_whence Flag for starting offset.  
7904 off\_t l\_start Relative offset in bytes.  
7905 off\_t l\_len Size; if 0 then until EOF.  
7906 pid\_t l\_pid Process ID of the process holding the lock; returned with F\_GETLK.

7907 The <fcntl.h> header shall define the **mode\_t**, **off\_t**, and **pid\_t** types as described in  
7908 <sys/types.h>.

7909 The following shall be declared as functions and may also be defined as macros. Function  
7910 prototypes shall be provided.

7911 int creat(const char \*, mode\_t);  
7912 int fcntl(int, int, ...);  
7913 int open(const char \*, int, ...);

```

7914 int openat(int, const char *, int, ...);
7915 ADV int posix_fadvise(int, off_t, off_t, int);
7916 int posix_fallocate(int, off_t, off_t);

```

7917 Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and  
 7918 <unistd.h>.

#### 7919 APPLICATION USAGE

7920 Although no existing implementation defines `AT_SYMLINK_FOLLOW` and  
 7921 `AT_SYMLINK_NOFOLLOW` as the same numeric value, POSIX.1-2008 does not prohibit that as  
 7922 the two constants are not used with the same interfaces.

#### 7923 RATIONALE

7924 While many of the symbolic constants introduced in the <fcntl.h> header do not strictly need to  
 7925 be used in `#if` preprocessor directives, widespread historic practice has defined them as macros  
 7926 that are usable in such constructs, and examination of existing applications has shown that they  
 7927 are occasionally used in such a way. Therefore it was decided to retain this requirement on an  
 7928 implementation in POSIX.1-2008.

#### 7929 FUTURE DIRECTIONS

7930 None.

#### 7931 SEE ALSO

7932 [<stdio.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

7933 XSH [creat\(\)](#), [exec](#), [fcntl\(\)](#), [futimens\(\)](#), [open\(\)](#), [posix\\_fadvise\(\)](#), [posix\\_fallocate\(\)](#), [posix\\_madvise\(\)](#)

#### 7934 CHANGE HISTORY

7935 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 7936 Issue 5

7937 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

#### 7938 Issue 6

7939 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 7940 • `O_DSYNC` and `O_RSYNC` are marked as part of the Synchronized Input and Output  
 7941 option.

7942 The following new requirements on POSIX implementations derive from alignment with the  
 7943 Single UNIX Specification:

- 7944 • The definition of the `mode_t`, `off_t`, and `pid_t` types is mandated.

7945 The `F_GETOWN` and `F_SETOWN` values are added for sockets.

7946 The `posix_fadvise()`, `posix_fallocate()`, and `posix_madvise()` functions are added for alignment with  
 7947 IEEE Std 1003.1d-1999.

7948 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for `posix_madvise()` to  
 7949 <sys/mman.h>.

7950 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for  
 7951 `posix_fadvise()` and `posix_fallocate()` to be large file-aware, using `off_t` instead of `size_t`.

#### 7952 Issue 7

7953 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the `O_TTY_INIT` flag.

7954 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
 7955 `FD_CLOEXEC` flag atomically at `open()`, and adding the `F_DUPFD_CLOEXEC` flag.

- 7956 The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API  
7957 Set Part 2.
- 7958 Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*,  
7959 *open()*, *openat()*, and *unlinkat()*.
- 7960 This reference page is clarified with respect to macros and symbolic constants.
- 7961 Changes are made related to support for finegrained timestamps.
- 7962 Changes are made to allow a directory to be opened for searching.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;fenv.h&gt;

## 7963 NAME

7964 fenv.h — floating-point environment

## 7965 SYNOPSIS

7966 #include &lt;fenv.h&gt;

## 7967 DESCRIPTION

7968 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 7969 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7970 volume of POSIX.1-2008 defers to the ISO C standard.

7971 The <fenv.h> header shall define the following data types through **typedef**:

7972 **fenv\_t** Represents the entire floating-point environment. The floating-point environment  
 7973 refers collectively to any floating-point status flags and control modes supported  
 7974 by the implementation.

7975 **feexcept\_t** Represents the floating-point status flags collectively, including any status the  
 7976 implementation associates with the flags. A floating-point status flag is a system  
 7977 variable whose value is set (but never cleared) when a floating-point exception is  
 7978 raised, which occurs as a side-effect of exceptional floating-point arithmetic to  
 7979 provide auxiliary information. A floating-point control mode is a system variable  
 7980 whose value may be set by the user to affect the subsequent behavior of floating-  
 7981 point arithmetic.

7982 The <fenv.h> header shall define each of the following macros if and only if the implementation  
 7983 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,  
 7984 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. The defined macros shall  
 7985 expand to integer constant expressions with values that are bitwise-distinct.

7986 FE\_DIVBYZERO  
 7987 FE\_INEXACT  
 7988 FE\_INVALID  
 7989 FE\_OVERFLOW  
 7990 FE\_UNDERFLOW

7991 MX If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be  
 7992 defined. Additional implementation-defined floating-point exceptions with macros beginning  
 7993 with FE\_ and an uppercase letter may also be specified by the implementation.

7994 The <fenv.h> header shall define the macro FE\_ALL\_EXCEPT as the bitwise-inclusive OR of all  
 7995 floating-point exception macros defined by the implementation, if any. If no such macros are  
 7996 defined, then the macro FE\_ALL\_EXCEPT shall be defined as zero.

7997 The <fenv.h> header shall define each of the following macros if and only if the implementation  
 7998 supports getting and setting the represented rounding direction by means of the *fegetround()*  
 7999 and *fesetround()* functions. The defined macros shall expand to integer constant expressions  
 8000 whose values are distinct non-negative values.

8001 FE\_DOWNWARD  
 8002 FE\_TONEAREST  
 8003 FE\_TOWARDZERO  
 8004 FE\_UPWARD

8005 MX If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be  
 8006 defined. Additional implementation-defined rounding directions with macros beginning with  
 8007 FE\_ and an uppercase letter may also be specified by the implementation.

8008 The <fenv.h> header shall define the following macro, which represents the default floating-  
 8009 point environment (that is, the one installed at program startup) and has type pointer to const-  
 8010 qualified `fenv_t`. It can be used as an argument to the functions within the <fenv.h> header that  
 8011 manage the floating-point environment.

8012 `FE_DFL_ENV`

8013 The following shall be declared as functions and may also be defined as macros. Function  
 8014 prototypes shall be provided.

```
8015 int feenableexcept(int);
8016 int fegetenv(fenv_t *);
8017 int fegetexceptflag(fexcept_t *, int);
8018 int fegetround(void);
8019 int feholdexcept(fenv_t *);
8020 int feraiseexcept(int);
8021 int fesetenv(const fenv_t *);
8022 int fesetexceptflag(const fexcept_t *, int);
8023 int fesetround(int);
8024 int fetestexcept(int);
8025 int feupdateenv(const fenv_t *);
```

8026 The `FENV_ACCESS` pragma provides a means to inform the implementation when an  
 8027 application might access the floating-point environment to test floating-point status flags or run  
 8028 under non-default floating-point control modes. The pragma shall occur either outside external  
 8029 declarations or preceding all explicit declarations and statements inside a compound statement.  
 8030 When outside external declarations, the pragma takes effect from its occurrence until another  
 8031 `FENV_ACCESS` pragma is encountered, or until the end of the translation unit. When inside a  
 8032 compound statement, the pragma takes effect from its occurrence until another `FENV_ACCESS`  
 8033 pragma is encountered (including within a nested compound statement), or until the end of the  
 8034 compound statement; at the end of a compound statement the state for the pragma is restored to  
 8035 its condition just before the compound statement. If this pragma is used in any other context, the  
 8036 behavior is undefined. If part of an application tests floating-point status flags, sets floating-  
 8037 point control modes, or runs under non-default mode settings, but was translated with the state  
 8038 for the `FENV_ACCESS` pragma off, the behavior is undefined. The default state (on or off) for  
 8039 the pragma is implementation-defined. (When execution passes from a part of the application  
 8040 translated with `FENV_ACCESS` off to a part translated with `FENV_ACCESS` on, the state of the  
 8041 floating-point status flags is unspecified and the floating-point control modes have their default  
 8042 settings.)

#### 8043 APPLICATION USAGE

8044 This header is designed to support the floating-point exception status flags and directed-  
 8045 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-  
 8046 point state information. Also it is designed to facilitate code portability among all systems.

8047 Certain application programming conventions support the intended model of use for the  
 8048 floating-point environment:

- 8049 • A function call does not alter its caller's floating-point control modes, clear its caller's  
 8050 floating-point status flags, nor depend on the state of its caller's floating-point status flags  
 8051 unless the function is so documented.
- 8052 • A function call is assumed to require default floating-point control modes, unless its  
 8053 documentation promises otherwise.

- 8054 • A function call is assumed to have the potential for raising floating-point exceptions,  
8055 unless its documentation promises otherwise.

8056 With these conventions, an application can safely assume default floating-point control modes  
8057 (or be unaware of them). The responsibilities associated with accessing the floating-point  
8058 environment fall on the application that does so explicitly.

8059 Even though the rounding direction macros may expand to constants corresponding to the  
8060 values of FLT\_ROUNDS, they are not required to do so.

8061 For example:

```
8062 #include <fenv.h>
8063 void f(double x)
8064 {
8065     #pragma STDC FENV_ACCESS ON
8066     void g(double);
8067     void h(double);
8068     /* ... */
8069     g(x + 1);
8070     h(x + 1);
8071     /* ... */
8072 }
```

8073 If the function *g()* might depend on status flags set as a side-effect of the first *x+1*, or if the  
8074 second *x+1* might depend on control modes set as a side-effect of the call to function *g()*, then  
8075 the application shall contain an appropriately placed invocation as follows:

```
8076 #pragma STDC FENV_ACCESS ON
```

## 8077 RATIONALE

### 8078 The `fexcept_t` Type

8079 `fexcept_t` does not have to be an integer type. Its values must be obtained by a call to  
8080 `fegetexceptflag()`, and cannot be created by logical operations from the exception macros. An  
8081 implementation might simply implement `fexcept_t` as an `int` and use the representations  
8082 reflected by the exception macros, but is not required to; other representations might contain  
8083 extra information about the exceptions. `fexcept_t` might be a `struct` with a member for each  
8084 exception (that might hold the address of the first or last floating-point instruction that caused  
8085 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an  
8086 `fexcept_t`, and so the user cannot inspect it.

### 8087 Exception and Rounding Macros

8088 Macros corresponding to unsupported modes and rounding directions are not defined by the  
8089 implementation and must not be defined by the application. An application might use `#ifdef`  
8090 test for this.

## 8091 FUTURE DIRECTIONS

8092 None.

## 8093 SEE ALSO

8094 XSH `feclearexcept()`, `fegetenv()`, `fegetexceptflag()`, `fegetround()`, `feholdexcept()`, `feraiseexcept()`,  
8095 `fetestexcept()`, `feupdateenv()`

8096 **CHANGE HISTORY**

8097 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8098 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,  
8099 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the  
8100 ISO/IEC 9899:1999 standard, Defect Report 202.

8101 **Issue 7**

8102 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied.

8103 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #36 is applied.

8104 SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied.

8105 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

8106 **NAME**

8107 float.h — floating types

8108 **SYNOPSIS**

8109 #include &lt;float.h&gt;

8110 **DESCRIPTION**

8111 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 8112 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8113 volume of POSIX.1-2008 defers to the ISO C standard.

8114 The characteristics of floating types are defined in terms of a model that describes a  
 8115 representation of floating-point numbers and values that provide information about an  
 8116 implementation's floating-point arithmetic.

8117 The following parameters are used to define the model for each floating-point type:

8118 *s* Sign ( $\pm 1$ ).8119 *b* Base or radix of exponent representation (an integer  $> 1$ ).8120 *e* Exponent (an integer between a minimum  $e_{\min}$  and a maximum  $e_{\max}$ ).8121 *p* Precision (the number of base-*b* digits in the significand).8122  $f_k$  Non-negative integers less than *b* (the significand digits).8123 A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8124 In addition to normalized floating-point numbers ( $f_1 > 0$  if  $x \neq 0$ ), floating types may be able to  
 8125 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ( $x \neq 0$ ,  
 8126  $e = e_{\min}$ ,  $f_1 = 0$ ) and unnormalized floating-point numbers ( $x \neq 0$ ,  $e > e_{\min}$ ,  $f_1 = 0$ ), and values that are  
 8127 not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-a-  
 8128 Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a  
 8129 floating-point exception; a *signaling NaN* generally raises a floating-point exception when  
 8130 occurring as an arithmetic operand.

8131 An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign,  
 8132 or may leave them unsigned. Wherever such values are unsigned, any requirement in  
 8133 POSIX.1-2008 to retrieve the sign shall produce an unspecified sign and any requirement to set  
 8134 the sign shall be ignored.

8135 The accuracy of the floating-point operations ('+', '-', '\*', '/') and of the functions in  
 8136 <math.h> and <complex.h> that return floating-point results is implementation-defined, as is  
 8137 the accuracy of the conversion between floating-point internal representations and string  
 8138 representations performed by the functions in <stdio.h>, <stdlib.h>, and <wchar.h>. The  
 8139 implementation may state that the accuracy is unknown.

8140 All integer values in the <float.h> header, except FLT\_ROUNDS, shall be constant expressions  
 8141 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.  
 8142 All except DECIMAL\_DIG, FLT\_EVAL\_METHOD, FLT\_RADIX, and FLT\_ROUNDS have  
 8143 separate names for all three floating-point types. The floating-point model representation is  
 8144 provided for all values except FLT\_EVAL\_METHOD and FLT\_ROUNDS.

8145 The rounding mode for floating-point addition is characterized by the implementation-defined

**<float.h>**

Headers

8146 value of FLT\_ROUNDS:

8147 -1 Indeterminable.

8148 0 Toward zero.

8149 1 To nearest.

8150 2 Toward positive infinity.

8151 3 Toward negative infinity.

8152 All other values for FLT\_ROUNDS characterize implementation-defined rounding behavior.

8153 The values of operations with floating operands and values subject to the usual arithmetic

8154 conversions and of floating constants are evaluated to a format whose range and precision may

8155 be greater than required by the type. The use of evaluation formats is characterized by the

8156 implementation-defined value of FLT\_EVAL\_METHOD:

8157 -1 Indeterminable.

8158 0 Evaluate all operations and constants just to the range and precision of the type.

8159 1 Evaluate operations and constants of type **float** and **double** to the range and precision of

8160 the **double** type; evaluate **long double** operations and constants to the range and precision

8161 of the **long double** type.

8162 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8163 All other negative values for FLT\_EVAL\_METHOD characterize implementation-defined

8164 behavior.

8165 The **<float.h>** header shall define the following values as constant expressions with

8166 implementation-defined values that are greater or equal in magnitude (absolute value) to those

8167 shown, with the same sign.

- 8168 • Radix of exponent representation,  $b$ .

8169 FLT\_RADIX 2

- 8170 • Number of base-FLT\_RADIX digits in the floating-point significand,  $p$ .

8171 FLT\_MANT\_DIG

8172 DBL\_MANT\_DIG

8173 LDBL\_MANT\_DIG

- 8174 • Number of decimal digits,  $n$ , such that any floating-point number in the widest supported
- 8175 floating type with  $p_{\max}$  radix  $b$  digits can be rounded to a floating-point number with  $n$
- 8176 decimal digits and back again without change to the value.

$$\left\{ \begin{array}{ll} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8177 DECIMAL\_DIG 10

## Headers

## &lt;float.h&gt;

- 8178 • Number of decimal digits,  $q$ , such that any floating-point number with  $q$  decimal digits can  
8179 be rounded into a floating-point number with  $p$  radix  $b$  digits and back again without  
8180 change to the  $q$  decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lfloor (p-1) \log_{10} b \rfloor & \text{otherwise} \end{cases}$$

8181 FLT\_DIG 6

8182 DBL\_DIG 10

8183 LDBL\_DIG 10

- 8184 • Minimum negative integer such that FLT\_RADIX raised to that power minus 1 is a  
8185 normalized floating-point number,  $e_{\min}$ .

8186 FLT\_MIN\_EXP

8187 DBL\_MIN\_EXP

8188 LDBL\_MIN\_EXP

- 8189 • Minimum negative integer such that 10 raised to that power is in the range of normalized  
8190 floating-point numbers.

$$\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$$

8191 FLT\_MIN\_10\_EXP -37

8192 DBL\_MIN\_10\_EXP -37

8193 LDBL\_MIN\_10\_EXP -37

- 8194 • Maximum integer such that FLT\_RADIX raised to that power minus 1 is a representable  
8195 finite floating-point number,  $e_{\max}$ .

8196 FLT\_MAX\_EXP

8197 DBL\_MAX\_EXP

8198 LDBL\_MAX\_EXP

- 8199 • Maximum integer such that 10 raised to that power is in the range of representable finite  
8200 floating-point numbers.

$$\left\lceil \log_{10} ((1 - b^{-p}) b^{e_{\max}}) \right\rceil$$

8201 FLT\_MAX\_10\_EXP +37

8202 DBL\_MAX\_10\_EXP +37

**<float.h>**

Headers

8203 LDBL\_MAX\_10\_EXP +37

8204 The **<float.h>** header shall define the following values as constant expressions with  
8205 implementation-defined values that are greater than or equal to those shown:

- 8206
- Maximum representable finite floating-point number.

$$(1 - b^{-p}) b^{e_{\max}}$$

8207 FLT\_MAX 1E+37

8208 DBL\_MAX 1E+37

8209 LDBL\_MAX 1E+37

8210 The **<float.h>** header shall define the following values as constant expressions with  
8211 implementation-defined (positive) values that are less than or equal to those shown:

- 8212
- The difference between 1 and the least value greater than 1 that is representable in the  
8213 given floating-point type,  $b^{1-p}$ .

8214 FLT\_EPSILON 1E-5

8215 DBL\_EPSILON 1E-9

8216 LDBL\_EPSILON 1E-9

- 8217
- Minimum normalized positive floating-point number,  $b^{e_{\min}-1}$ .

8218 FLT\_MIN 1E-37

8219 DBL\_MIN 1E-37

8220 LDBL\_MIN 1E-37

8221 **APPLICATION USAGE**

8222 None.

8223 **RATIONALE**

8224 None.

8225 **FUTURE DIRECTIONS**

8226 None.

8227 **SEE ALSO**8228 [<complex.h>](#), [<math.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<wchar.h>](#)8229 **CHANGE HISTORY**

8230 First released in Issue 4. Derived from the ISO C standard.

8231 **Issue 6**8232 The description of the operations with floating-point values is updated for alignment with the  
8233 ISO/IEC 9899:1999 standard.8234 **Issue 7**8235 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) and #5  
8236 (SD5-XBD-ERN-51) are applied.

## Headers

## &lt;fmtmsg.h&gt;

8237 **NAME**8238 `fmtmsg.h` — message display structures8239 **SYNOPSIS**8240 XSI `#include <fmtmsg.h>`8241 **DESCRIPTION**

8242 The &lt;fmtmsg.h&gt; header shall define the following symbolic constants:

8243	MM_HARD	Source of the condition is hardware.
8244	MM_SOFT	Source of the condition is software.
8245	MM_FIRM	Source of the condition is firmware.
8246	MM_APPL	Condition detected by application.
8247	MM_UTIL	Condition detected by utility.
8248	MM_OPSYS	Condition detected by operating system.
8249	MM_RECOVER	Recoverable error.
8250	MM_NRECOV	Non-recoverable error.
8251	MM_HALT	Error causing application to halt.
8252	MM_ERROR	Application has encountered a non-fatal fault.
8253	MM_WARNING	Application has detected unusual non-error condition.
8254	MM_INFO	Informative message.
8255	MM_NOSEV	No severity level provided for the message.
8256	MM_PRINT	Display message on standard error.
8257	MM_CONSOLE	Display message on system console.

8258 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The  
 8259 <fmtmsg.h> header shall define the symbolic constants in the **Identifier** column, which shall  
 8260 have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8268 The <fmtmsg.h> header shall also define the following symbolic constants for use as return  
 8269 values for *fmtmsg()*:

8270	MM_OK	The function succeeded.
8271	MM_NOTOK	The function failed completely.
8272	MM_NOMSG	The function was unable to generate a message on standard error, but 8273 otherwise succeeded.

## &lt;fmtmsg.h&gt;

Headers

8274 MM\_NOCON The function was unable to generate a console message, but otherwise  
8275 succeeded.

8276 The following shall be declared as a function and may also be defined as a macro. A function  
8277 prototype shall be provided.

```
8278 int fmtmsg(long, const char *, int,  
8279           const char *, const char *, const char *);
```

**8280 APPLICATION USAGE**

8281 None.

**8282 RATIONALE**

8283 None.

**8284 FUTURE DIRECTIONS**

8285 None.

**8286 SEE ALSO**

8287 XSH *fmtmsg()*

**8288 CHANGE HISTORY**

8289 First released in Issue 4, Version 2.

**8290 Issue 7**

8291 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

8292 **NAME**

8293 fnmatch.h — filename-matching types

8294 **SYNOPSIS**

8295 #include &lt;fnmatch.h&gt;

8296 **DESCRIPTION**

8297 The &lt;fnmatch.h&gt; header shall define the following symbolic constants:

8298 FNM\_NOMATCH The string does not match the specified pattern.

8299 FNM\_PATHNAME <slash> in *string* only matches <slash> in *pattern*.8300 FNM\_PERIOD Leading <period> in *string* must be exactly matched by <period> in  
8301 *pattern*.

8302 FNM\_NOESCAPE Disable backslash escaping.

8303 The following shall be declared as a function and may also be defined as a macro. A function  
8304 prototype shall be provided.

8305 int fnmatch(const char \*, const char \*, int);

8306 **APPLICATION USAGE**

8307 None.

8308 **RATIONALE**

8309 None.

8310 **FUTURE DIRECTIONS**

8311 None.

8312 **SEE ALSO**8313 XSH *fnmatch()*8314 **CHANGE HISTORY**

8315 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8316 **Issue 6**

8317 The FNM\_NOSYS constant is marked obsolescent.

8318 **Issue 7**

8319 The obsolescent FNM\_NOSYS constant is removed.

8320 This reference page is clarified with respect to macros and symbolic constants.

## &lt;ftw.h&gt;

Headers

## 8321 NAME

8322 ftw.h — file tree traversal

## 8323 SYNOPSIS

8324 xSI `#include <ftw.h>`

## 8325 DESCRIPTION

8326 The <ftw.h> header shall define the **FTW** structure, which shall include at least the following  
8327 members:8328 `int base`  
8329 `int level`8330 The <ftw.h> header shall define the following symbolic constants for use as values of the third  
8331 argument to the application-supplied function that is passed as the second argument to *ftw()*  
8332 and *nftw()*:8333 **FTW\_F** File.  
8334 **FTW\_D** Directory.  
8335 **FTW\_DNR** Directory without read permission.  
8336 **FTW\_DP** Directory with subdirectories visited.  
8337 **FTW\_NS** Unknown type; *stat()* failed.  
8338 **FTW\_SL** Symbolic link.  
8339 **FTW\_SLN** Symbolic link that names a nonexistent file.8340 The <ftw.h> header shall define the following symbolic constants for use as values of the fourth  
8341 argument to *nftw()*:8342 **FTW\_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows  
8343 links but does not walk down any path that crosses itself.  
8344 **FTW\_MOUNT** The walk does not cross a mount point.  
8345 **FTW\_DEPTH** All subdirectories are visited before the directory itself.  
8346 **FTW\_CHDIR** The walk changes to each directory before reading it.8347 The following shall be declared as functions and may also be defined as macros. Function  
8348 prototypes shall be provided.8349 OB `int ftw(const char *, int (*)(const char *, const struct stat *,`  
8350 `int), int);`  
8351 `int nftw(const char *, int (*)(const char *, const struct stat *,`  
8352 `int, struct FTW *), int, int);`8353 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st\_mode* and the  
8354 file type test macros as described in <sys/stat.h>.

8355 Inclusion of the &lt;ftw.h&gt; header may also make visible all symbols from &lt;sys/stat.h&gt;.

## Headers

&lt;ftw.h&gt;

- 8356 **APPLICATION USAGE**  
8357       None.
- 8358 **RATIONALE**  
8359       None.
- 8360 **FUTURE DIRECTIONS**  
8361       None.
- 8362 **SEE ALSO**  
8363       [<sys/stat.h>](#)  
8364       XSH *ftw()*, *nftw()*
- 8365 **CHANGE HISTORY**  
8366       First released in Issue 1. Derived from Issue 1 of the SVID.
- 8367 **Issue 5**  
8368       A description of FTW\_DP is added.
- 8369 **Issue 7**  
8370       The *ftw()* function is marked obsolescent.  
8371       This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<glob.h>**

Headers

8372 **NAME**

8373 glob.h — pathname pattern-matching types

8374 **SYNOPSIS**

8375 #include &lt;glob.h&gt;

8376 **DESCRIPTION**8377 The **<glob.h>** header shall define the structures and symbolic constants used by the *glob()*  
8378 function.8379 The **<glob.h>** header shall define the **glob\_t** structure type, which shall include at least the  
8380 following members:

8381 `size_t gl_pathc` Count of paths matched by *pattern*.  
 8382 `char **gl_pathv` Pointer to a list of matched pathnames.  
 8383 `size_t gl_offs` Slots to reserve at the beginning of *gl\_pathv*.

8384 The **<glob.h>** header shall define the **size\_t** type as described in **<sys/types.h>**.8385 The **<glob.h>** header shall define the following symbolic constants as values for the *flags*  
8386 argument:

8387 **GLOB\_APPEND** Append generated pathnames to those previously obtained.  
 8388 **GLOB\_DOOFFS** Specify how many null pointers to add to the beginning of *gl\_pathv*.  
 8389 **GLOB\_ERR** Cause *glob()* to return on error.  
 8390 **GLOB\_MARK** Each pathname that is a directory that matches *pattern* has a <slash>  
 8391 appended.  
 8392 **GLOB\_NOCHECK** If *pattern* does not match any pathname, then return a list consisting of  
 8393 only *pattern*.  
 8394 **GLOB\_NOESCAPE** Disable backslash escaping.  
 8395 **GLOB\_NOSORT** Do not sort the pathnames returned.  
 8396 The **<glob.h>** header shall define the following symbolic constants as error return values:  
 8397 **GLOB\_ABORTED** The scan was stopped because **GLOB\_ERR** was set or *(\*errfunc)()*  
 8398 returned non-zero.  
 8399 **GLOB\_NOMATCH** The pattern does not match any existing pathname, and  
 8400 **GLOB\_NOCHECK** was not set in *flags*.  
 8401 **GLOB\_NOSPACE** An attempt to allocate memory failed.

8402 The following shall be declared as functions and may also be defined as macros. Function  
8403 prototypes shall be provided.

8404 `int glob(const char *restrict, int, int (*)(const char *, int),`  
 8405 `glob_t *restrict);`  
 8406 `void globfree(glob_t *);`

8407 **APPLICATION USAGE**

8408 None.

8409 **RATIONALE**

8410 None.

8411 **FUTURE DIRECTIONS**

8412 None.

8413 **SEE ALSO**

8414 &lt;sys/types.h&gt;

8415 XSH *glob()*8416 **CHANGE HISTORY**

8417 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8418 **Issue 6**8419 The **restrict** keyword is added to the prototype for *glob()*.

8420 The GLOB\_NOSYS constant is marked obsolescent.

8421 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()*  
8422 prototype definition by removing the **restrict** qualifier from the function pointer argument.8423 **Issue 7**8424 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t**

8425 The obsolescent GLOB\_NOSYS constant is removed.

8426 This reference page is clarified with respect to macros and symbolic constants.

**<grp.h>**

Headers

8427 **NAME**

8428           grp.h — group structure

8429 **SYNOPSIS**

8430           #include &lt;grp.h&gt;

8431 **DESCRIPTION**8432           The **<grp.h>** header shall declare the **group** structure, which shall include the following  
8433 members:

8434           char     \*gr\_name   The name of the group.  
 8435           gid\_t    gr\_gid     Numerical group ID.  
 8436           char    \*\*gr\_mem    Pointer to a null-terminated array of character  
 8437                               pointers to member names.

8438           The **<grp.h>** header shall define the **gid\_t** and **size\_t** types as described in **<sys/types.h>**.8439           The following shall be declared as functions and may also be defined as macros. Function  
8440 prototypes shall be provided.

```

8441           void            endgrent(void);
8442           struct group    *getgrent(void);
8443           struct group    *getgrgid(gid_t);
8444           int             getgrgid_r(gid_t, struct group *, char *,
8445                           size_t, struct group **);
8446           struct group    *getgrnam(const char *);
8447           int             getgrnam_r(const char *, struct group *, char *,
8448                           size_t, struct group **);
8449 XSI        void            setgrent(void);

```

8450 **APPLICATION USAGE**

8451           None.

8452 **RATIONALE**

8453           None.

8454 **FUTURE DIRECTIONS**

8455           None.

8456 **SEE ALSO**8457           [<sys/types.h>](#)8458           XSH [endgrent\(\)](#), [getgrgid\(\)](#), [getgrnam\(\)](#)8459 **CHANGE HISTORY**

8460           First released in Issue 1.

8461 **Issue 5**

8462           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8463 **Issue 6**8464           The following new requirements on POSIX implementations derive from alignment with the  
8465 Single UNIX Specification:

- 8466           • The definition of **gid\_t** is mandated.
- 8467           • The [getgrgid\\_r\(\)](#) and [getgrnam\\_r\(\)](#) functions are marked as part of the Thread-Safe  
8468 Functions option.

*Headers***<grp.h>**8469  
8470**Issue 7**SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size\_t** type.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<iconv.h>**

Headers

8471 **NAME**

8472 iconv.h — codeset conversion facility

8473 **SYNOPSIS**

8474 #include &lt;iconv.h&gt;

8475 **DESCRIPTION**

8476 The &lt;iconv.h&gt; header shall define the following types:

8477 **iconv\_t** Identifies the conversion from one codeset to another.8478 **size\_t** As described in <sys/types.h>.8479 The following shall be declared as functions and may also be defined as macros. Function  
8480 prototypes shall be provided.

```

8481 size_t iconv(iconv_t, char **restrict, size_t *restrict,
8482             char **restrict, size_t *restrict);
8483 int iconv_close(iconv_t);
8484 iconv_t iconv_open(const char *, const char *);

```

8485 **APPLICATION USAGE**

8486 None.

8487 **RATIONALE**

8488 None.

8489 **FUTURE DIRECTIONS**

8490 None.

8491 **SEE ALSO**

8492 &lt;sys/types.h&gt;

8493 XSH *iconv()*, *iconv\_close()*, *iconv\_open()*8494 **CHANGE HISTORY**

8495 First released in Issue 4.

8496 **Issue 6**8497 The **restrict** keyword is added to the prototype for *iconv()*.8498 **Issue 7**8499 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

8500 The &lt;iconv.h&gt; header is moved from the XSI option to the Base.

## Headers

## &lt;inttypes.h&gt;

## 8501 NAME

8502 inttypes.h — fixed size integer types

## 8503 SYNOPSIS

8504 #include &lt;inttypes.h&gt;

## 8505 DESCRIPTION

8506 CX Some of the functionality described on this reference page extends the ISO C standard.  
 8507 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 8508 enable the visibility of these symbols in this header.

8509 The &lt;inttypes.h&gt; header shall include the &lt;stdint.h&gt; header.

8510 The &lt;inttypes.h&gt; header shall define at least the following type:

8511 **imaxdiv\_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8512 The <inttypes.h> header shall define the following macros. Each expands to a character string  
 8513 literal containing a conversion specifier, possibly modified by a length modifier, suitable for use  
 8514 within the *format* argument of a formatted input/output function when converting the  
 8515 corresponding integer type. These macros have the general form of PRI (character string literals  
 8516 for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the  
 8517 *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a  
 8518 name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the  
 8519 width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format  
 8520 string to print the value of an integer of type *int\_fast32\_t*.

8521 The *fprintf()* macros for signed integers are:

8522	<i>PRIdN</i>	<i>PRIdLEASTN</i>	<i>PRIdFASTN</i>	<i>PRIdMAX</i>	<i>PRIdPTR</i>
8523	<i>PRiN</i>	<i>PRiLEASTN</i>	<i>PRiFASTN</i>	<i>PRiMAX</i>	<i>PRiPTR</i>

8524 The *fprintf()* macros for unsigned integers are:

8525	<i>PRIoN</i>	<i>PRIoLEASTN</i>	<i>PRIoFASTN</i>	<i>PRIoMAX</i>	<i>PRIoPTR</i>
8526	<i>PRiuN</i>	<i>PRiuLEASTN</i>	<i>PRiuFASTN</i>	<i>PRiuMAX</i>	<i>PRiuPTR</i>
8527	<i>PRIxN</i>	<i>PRIxLEASTN</i>	<i>PRIxFASTN</i>	<i>PRIxMAX</i>	<i>PRIxPTR</i>
8528	<i>PRIXN</i>	<i>PRIXLEASTN</i>	<i>PRIXFASTN</i>	<i>PRIXMAX</i>	<i>PRIXPTR</i>

8529 The *fscanf()* macros for signed integers are:

8530	<i>SCNdN</i>	<i>SCNdLEASTN</i>	<i>SCNdFASTN</i>	<i>SCNdMAX</i>	<i>SCNdPTR</i>
8531	<i>SCNiN</i>	<i>SCNiLEASTN</i>	<i>SCNiFASTN</i>	<i>SCNiMAX</i>	<i>SCNiPTR</i>

8532 The *fscanf()* macros for unsigned integers are:

8533	<i>SCNoN</i>	<i>SCNoLEASTN</i>	<i>SCNoFASTN</i>	<i>SCNoMAX</i>	<i>SCNoPTR</i>
8534	<i>SCNuN</i>	<i>SCNuLEASTN</i>	<i>SCNuFASTN</i>	<i>SCNuMAX</i>	<i>SCNuPTR</i>
8535	<i>SCNxN</i>	<i>SCNxLEASTN</i>	<i>SCNxFASTN</i>	<i>SCNxMAX</i>	<i>SCNxPTR</i>

8536 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and  
 8537 *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be  
 8538 defined unless the implementation does not have a suitable modifier for the type.

8539 The following shall be declared as functions and may also be defined as macros. Function  
 8540 prototypes shall be provided.

```
8541 intmax_t imaxabs(intmax_t);
8542 imaxdiv_t imaxdiv(intmax_t, intmax_t);
```

**<inttypes.h>**

Headers

```

8543     intmax_t  strtoumax(const char *restrict, char **restrict, int);
8544     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8545     intmax_t  wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
8546     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

**EXAMPLES**

```

8547     #include <inttypes.h>
8548     #include <wchar.h>
8549     int main(void)
8550     {
8551         uintmax_t i = UINTMAX_MAX; // This type always exists.
8552         wprintf(L"The largest integer value is %020"
8553             PRIxMAX "\n", i);
8554         return 0;
8555     }
8556

```

**APPLICATION USAGE**

8557 The purpose of **<inttypes.h>** is to provide a set of integer types whose definitions are consistent  
8558 across machines and independent of operating systems and other implementation  
8559 idiosyncrasies. It defines, through **typedef**, integer types of various sizes. Implementations are  
8560 free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent  
8561 use of this header will greatly increase the portability of applications across platforms.

**RATIONALE**

8562 The ISO/IEC 9899:1990 standard specified that the language should support four signed and  
8563 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on  
8564 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int**  
8565 and not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits  
8566 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to  
8567 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems  
8568 for users who migrate from one system to another which assigns different sizes to integer types,  
8569 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.  
8570 The need for defining an extended integer type increased with the introduction of 64-bit  
8571 systems.

**FUTURE DIRECTIONS**

8572 Macro names beginning with PRI or SCN followed by any lowercase letter or 'X' may be added  
8573 to the macros defined in the **<inttypes.h>** header.

**SEE ALSO**

8574 XSH Section 2.2 (on page 468), *imaxabs()*, *imaxdiv()*, *strtoumax()*, *wcstoumax()*

**CHANGE HISTORY**

8575 First released in Issue 5.

**Issue 6**

8576 The Open Group Base Resolution bwg97-006 is applied.

8577 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

## Headers

## &lt;iso646.h&gt;

8584 **NAME**

8585           iso646.h — alternative spellings

8586 **SYNOPSIS**

8587           #include &lt;iso646.h&gt;

8588 **DESCRIPTION**

8589 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 8590 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8591 volume of POSIX.1-2008 defers to the ISO C standard.

8592           The <iso646.h> header shall define the following eleven macros (on the left) that expand to the  
 8593 corresponding tokens (on the right):

8594           and           &amp;&amp;

8595           and\_eq        &amp;=

8596           bitand       &amp;

8597           bitor         |

8598           compl        ~

8599           not           !

8600           not\_eq       !=

8601           or           ||

8602           or\_eq       |=

8603           xor          ^

8604           xor\_eq       ^=

8605 **APPLICATION USAGE**

8606           None.

8607 **RATIONALE**

8608           None.

8609 **FUTURE DIRECTIONS**

8610           None.

8611 **SEE ALSO**

8612           None.

8613 **CHANGE HISTORY**

8614           First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

<langinfo.h>

8615 **NAME**  
 8616 langinfo.h — language information constants

8617 **SYNOPSIS**  
 8618 #include <langinfo.h>

8619 **DESCRIPTION**  
 8620 The <langinfo.h> header shall define the symbolic constants used to identify items of *langinfo*  
 8621 data (see *nl\_langinfo()*).

8622 The <langinfo.h> header shall define the **locale\_t** type as described in <locale.h>.

8623 The <langinfo.h> header shall define the **nl\_item** type as described in <nl/types.h>.

8624 The <langinfo.h> header shall define the following symbolic constants with type **nl\_item**. The  
 8625 entries under **Category** indicate in which *setlocale()* category each item is defined.

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante-meridiem affix.
PM_STR	LC_TIME	Post-meridiem affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.
ABMON_1	LC_TIME	Abbreviated name of the first month.

	Constant	Category	Meaning
8663			
8664	ABMON_2	LC_TIME	Abbreviated name of the second month.
8665	ABMON_3	LC_TIME	Abbreviated name of the third month.
8666	ABMON_4	LC_TIME	Abbreviated name of the fourth month.
8667	ABMON_5	LC_TIME	Abbreviated name of the fifth month.
8668	ABMON_6	LC_TIME	Abbreviated name of the sixth month.
8669	ABMON_7	LC_TIME	Abbreviated name of the seventh month.
8670	ABMON_8	LC_TIME	Abbreviated name of the eighth month.
8671	ABMON_9	LC_TIME	Abbreviated name of the ninth month.
8672	ABMON_10	LC_TIME	Abbreviated name of the tenth month.
8673	ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
8674	ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
8675	ERA	LC_TIME	Era description segments.
8676	ERA_D_FMT	LC_TIME	Era date format string.
8677	ERA_D_T_FMT	LC_TIME	Era date and time format string.
8678	ERA_T_FMT	LC_TIME	Era time format string.
8679	ALT_DIGITS	LC_TIME	Alternative symbols for digits.
8680	RADIXCHAR	LC_NUMERIC	Radix character.
8681	THOUSEP	LC_NUMERIC	Separator for thousands.
8682	YESEXPR	LC_MESSAGES	Affirmative response expression.
8683	NOEXPR	LC_MESSAGES	Negative response expression.
8684	CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").
8685			
8686			
8687			
8688			

8689 If the locale's values for **p\_cs\_precedes** and **n\_cs\_precedes** do not match, the value of  
8690 *nl\_langinfo*(CRNCYSTR) and *nl\_langinfo\_l*(CRNCYSTR,loc) is unspecified.

8691 The following shall be declared as a function and may also be defined as a macro. A function  
8692 prototype shall be provided:

```
8693 char *nl_langinfo(nl_item);
8694 char *nl_langinfo_l(nl_item, locale_t);
```

8695 Inclusion of the <langinfo.h> header may also make visible all symbols from <nl\_types.h>.

## 8696 APPLICATION USAGE

8697 Wherever possible, users are advised to use functions compatible with those in the ISO C  
8698 standard to access items of *langinfo* data. In particular, the *strftime*() function should be used to  
8699 access date and time information defined in category *LC\_TIME*. The *localeconv*() function  
8700 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and  
8701 *CRNCYSTR*.

## 8702 RATIONALE

8703 None.

## 8704 FUTURE DIRECTIONS

8705 None.

## 8706 SEE ALSO

8707 [Chapter 7](#) (on page 135), <locale.h>, <nl\_types.h>

8708 XSH *nl\_langinfo*(), *localeconv*(), *strfmon*(), *strftime*()

## &lt;langinfo.h&gt;

Headers

8709 **CHANGE HISTORY**

8710 First released in Issue 2.

8711 **Issue 5**

8712 The constants YESSTR and NOSTR are marked LEGACY.

8713 **Issue 6**

8714 The constants YESSTR and NOSTR are removed.

8715 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to  
8716 the "Meaning" column entry for the CRNCYSTR constant. This change is to accommodate  
8717 historic practice.

8718 **Issue 7**

8719 The &lt;langinfo.h&gt; header is moved from the XSI option to the Base.

8720 The *nl\_langinfo\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
8721 API Set Part 4.

8722 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
8723 for the *locale\_t* type is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

8724 **NAME**

8725 libgen.h — definitions for pattern matching functions

8726 **SYNOPSIS**8727 XSI `#include <libgen.h>`8728 **DESCRIPTION**8729 The following shall be declared as functions and may also be defined as macros. Function  
8730 prototypes shall be provided.8731 `char *basename(char *);`8732 `char *dirname(char *);`8733 **APPLICATION USAGE**

8734 None.

8735 **RATIONALE**

8736 None.

8737 **FUTURE DIRECTIONS**

8738 None.

8739 **SEE ALSO**8740 XSH *basename()*, *dirname()*8741 **CHANGE HISTORY**

8742 First released in Issue 4, Version 2.

8743 **Issue 5**8744 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first  
8745 argument is of type **char \*** rather than **const char \***.8746 **Issue 6**8747 The `__loc1` symbol and the *regcmp()* and *regex()* functions are removed.

**<limits.h>**

Headers

8748 **NAME**

8749 limits.h — implementation-defined constants

8750 **SYNOPSIS**

8751 #include &lt;limits.h&gt;

8752 **DESCRIPTION**

8753 CX Some of the functionality described on this reference page extends the ISO C standard.  
 8754 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 8755 enable the visibility of these symbols in this header.

8756 Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such  
 8757 symbols are not shown as CX shaded.

8758 The **<limits.h>** header shall define macros and symbolic constants for various limits. Different  
 8759 categories of limits are described below, representing various limits on resources that the  
 8760 implementation imposes on applications. All macros and symbolic constants defined in this  
 8761 header shall be suitable for use in **#if** preprocessing directives.

8762 Implementations may choose any appropriate value for each limit, provided it is not more  
 8763 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names  
 8764 beginning with **\_POSIX** may be found in **<unistd.h>**.

8765 Applications should not assume any particular value for a limit. To achieve maximum  
 8766 portability, an application should not require more resource than the Minimum Acceptable  
 8767 Value quantity. However, an application wishing to avail itself of the full amount of a resource  
 8768 available on an implementation may make use of the value given in **<limits.h>** on that  
 8769 particular implementation, by using the macros and symbolic constants listed below. It should  
 8770 be noted, however, that many of the listed limits are not invariant, and at runtime, the value of  
 8771 the limit may differ from those given in this header, for the following reasons:

- 8772 • The limit is pathname-dependent.
- 8773 • The limit differs between the compile and runtime machines.

8774 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to  
 8775 determine the actual value of a limit at runtime.

8776 The items in the list ending in **\_MIN** give the most negative values that the mathematical types  
 8777 are guaranteed to be capable of representing. Numbers of a more negative value may be  
 8778 supported on some implementations, as indicated by the **<limits.h>** header on the  
 8779 implementation, but applications requiring such numbers are not guaranteed to be portable to  
 8780 all implementations. For positive constants ending in **\_MIN**, this indicates the minimum  
 8781 acceptable value.

8782 **Runtime Invariant Values (Possibly Indeterminate)**

8783 A definition of one of the symbolic constants in the following list shall be omitted from  
 8784 **<limits.h>** on specific implementations where the corresponding value is equal to or greater  
 8785 than the stated minimum, but is unspecified.

8786 This indetermination might depend on the amount of available memory space on a specific  
 8787 instance of a specific implementation. The actual value supported by a specific instance shall be  
 8788 provided by the *sysconf()* function.

8789 {AIO\_LISTIO\_MAX}

8790 Maximum number of I/O operations in a single list I/O call supported by the  
 8791 implementation.

8792 Minimum Acceptable Value: {\_POSIX\_AIO\_LISTIO\_MAX}

## Headers

## &lt;limits.h&gt;

8793		{AIO_MAX}
8794		Maximum number of outstanding asynchronous I/O operations supported by the implementation.
8795		Minimum Acceptable Value: {_POSIX_AIO_MAX}
8796		
8797		{AIO_PRIO_DELTA_MAX}
8798		The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
8799		Minimum Acceptable Value: 0
8800		
8801		{ARG_MAX}
8802		Maximum length of argument to the <i>exec</i> functions including environment data.
8803		Minimum Acceptable Value: {_POSIX_ARG_MAX}
8804		{ATEXIT_MAX}
8805		Maximum number of functions that may be registered with <i>atexit</i> ().
8806		Minimum Acceptable Value: 32
8807		{CHILD_MAX}
8808		Maximum number of simultaneous processes per real user ID.
8809		Minimum Acceptable Value: {_POSIX_CHILD_MAX}
8810		{DELAYTIMER_MAX}
8811		Maximum number of timer expiration overruns.
8812		Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}
8813		{HOST_NAME_MAX}
8814		Maximum length of a host name (not including the terminating null) as returned from the <i>gethostname</i> () function.
8815		Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}
8816		
8817	XSI	{IOV_MAX}
8818		Maximum number of <i>iovec</i> structures that one process has available for use with <i>readv</i> () or <i>writev</i> ().
8819		Minimum Acceptable Value: {_XOPEN_IOV_MAX}
8820		
8821		{LOGIN_NAME_MAX}
8822		Maximum length of a login name.
8823		Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}
8824	MSG	{MQ_OPEN_MAX}
8825		The maximum number of open message queue descriptors a process may hold.
8826		Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}
8827	MSG	{MQ_PRIO_MAX}
8828		The maximum number of message priorities supported by the implementation.
8829		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8830		{OPEN_MAX}
8831		A value one greater than the maximum value that the system may assign to a newly-created file descriptor.
8832		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8833		
8834		{PAGESIZE}
8835		Size in bytes of a page.
8836		Minimum Acceptable Value: 1

8837	XSI	<b>{PAGE_SIZE}</b>
8838		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8839		defined with the same value.
8840		<b>{PTHREAD_DESTRUCTOR_ITERATIONS}</b>
8841		Maximum number of attempts made to destroy a thread's thread-specific data values on
8842		thread exit.
8843		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8844		<b>{PTHREAD_KEYS_MAX}</b>
8845		Maximum number of data keys that can be created by a process.
8846		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8847		<b>{PTHREAD_STACK_MIN}</b>
8848		Minimum size in bytes of thread stack storage.
8849		Minimum Acceptable Value: 0
8850		<b>{PTHREAD_THREADS_MAX}</b>
8851		Maximum number of threads that can be created per process.
8852		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8853		<b>{RE_DUP_MAX}</b>
8854		Maximum number of repeated occurrences of a BRE or ERE interval expression; see <a href="#">Section</a>
8855		<a href="#">9.3.6</a> (on page 186) and <a href="#">Section 9.4.6</a> (on page 189).
8856		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8857		<b>{RTSIG_MAX}</b>
8858		Maximum number of realtime signals reserved for application use in this implementation.
8859		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8860		<b>{SEM_NSEMS_MAX}</b>
8861		Maximum number of semaphores that a process may have.
8862		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8863		<b>{SEM_VALUE_MAX}</b>
8864		The maximum value a semaphore may have.
8865		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8866		<b>{SIGQUEUE_MAX}</b>
8867		Maximum number of queued signals that a process may send and have pending at the
8868		receiver(s) at any time.
8869		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8870	SSI TSP	<b>{SS_REPL_MAX}</b>
8871		The maximum number of replenishment operations that may be simultaneously pending
8872		for a particular sporadic server scheduler.
8873		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
8874		<b>{STREAM_MAX}</b>
8875		Maximum number of streams that one process can have open at one time. If defined, it has
8876		the same value as {FOPEN_MAX} (see <stdio.h>).
8877		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
8878		<b>{SYMLOOP_MAX}</b>
8879		Maximum number of symbolic links that can be reliably traversed in the resolution of a
8880		pathname in the absence of a loop.
8881		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

8882 {TIMER\_MAX}  
 8883 Maximum number of timers per process supported by the implementation.  
 8884 Minimum Acceptable Value: {\_POSIX\_TIMER\_MAX}

8885 OB TRC {TRACE\_EVENT\_NAME\_MAX}  
 8886 Maximum length of the trace event name (not including the terminating null).  
 8887 Minimum Acceptable Value: {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}

8888 OB TRC {TRACE\_NAME\_MAX}  
 8889 Maximum length of the trace generation version string or of the trace stream name (not  
 8890 including the terminating null).  
 8891 Minimum Acceptable Value: {\_POSIX\_TRACE\_NAME\_MAX}

8892 OB TRC {TRACE\_SYS\_MAX}  
 8893 Maximum number of trace streams that may simultaneously exist in the system.  
 8894 Minimum Acceptable Value: {\_POSIX\_TRACE\_SYS\_MAX}

8895 OB TRC {TRACE\_USER\_EVENT\_MAX}  
 8896 Maximum number of user trace event type identifiers that may simultaneously exist in a  
 8897 traced process, including the predefined user trace event  
 8898 POSIX\_TRACE\_UNNAMED\_USER\_EVENT.  
 8899 Minimum Acceptable Value: {\_POSIX\_TRACE\_USER\_EVENT\_MAX}

8900 {TTY\_NAME\_MAX}  
 8901 Maximum length of terminal device name.  
 8902 Minimum Acceptable Value: {\_POSIX\_TTY\_NAME\_MAX}

8903 {TZNAME\_MAX}  
 8904 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).  
 8905 Minimum Acceptable Value: {\_POSIX\_TZNAME\_MAX}

8906 **Note:** The length given by {TZNAME\_MAX} does not include the quoting characters mentioned in  
 8907 [Section 8.3](#) (on page 177).

#### 8908 Pathname Variable Values

8909 The values in the following list may be constants within an implementation or may vary from  
 8910 one pathname to another. For example, file systems or directories may have different  
 8911 characteristics.

8912 A definition of one of the symbolic constants in the following list shall be omitted from the  
 8913 <limits.h> header on specific implementations where the corresponding value is equal to or  
 8914 greater than the stated minimum, but where the value can vary depending on the file to which it  
 8915 is applied. The actual value supported for a specific pathname shall be provided by the  
 8916 *pathconf()* function.

8917 {FILESIZEBITS}  
 8918 Minimum number of bits needed to represent, as a signed integer value, the maximum size  
 8919 of a regular file allowed in the specified directory.  
 8920 Minimum Acceptable Value: 32

8921 {LINK\_MAX}  
 8922 Maximum number of links to a single file.  
 8923 Minimum Acceptable Value: {\_POSIX\_LINK\_MAX}

8924 {MAX\_CANON}  
 8925 Maximum number of bytes in a terminal canonical input line.  
 8926 Minimum Acceptable Value: {\_POSIX\_MAX\_CANON}

8927		{MAX_INPUT}
8928		Minimum number of bytes for which space is available in a terminal input queue; therefore,
8929		the maximum number of bytes a conforming application may require to be typed as input
8930		before reading them.
8931		Minimum Acceptable Value: {_POSIX_MAX_INPUT}
8932		{NAME_MAX}
8933		Maximum number of bytes in a filename (not including the terminating null).
8934		Minimum Acceptable Value: {_POSIX_NAME_MAX}
8935	XSI	Minimum Acceptable Value: {_XOPEN_NAME_MAX}
8936		{PATH_MAX}
8937		Maximum number of bytes the implementation will store as a pathname in a user-supplied
8938		buffer of unspecified size, including the terminating null character. Minimum number the
8939		implementation will accept as the maximum number of bytes in a pathname.
8940		Minimum Acceptable Value: {_POSIX_PATH_MAX}
8941	XSI	Minimum Acceptable Value: {_XOPEN_PATH_MAX}
8942		{PIPE_BUF}
8943		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
8944		Minimum Acceptable Value: {_POSIX_PIPE_BUF}
8945	ADV	{POSIX_ALLOC_SIZE_MIN}
8946		Minimum number of bytes of storage actually allocated for any portion of a file.
8947		Minimum Acceptable Value: Not specified.
8948	ADV	{POSIX_REC_INCR_XFER_SIZE}
8949		Recommended increment for file transfer sizes between the
8950		{POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
8951		Minimum Acceptable Value: Not specified.
8952	ADV	{POSIX_REC_MAX_XFER_SIZE}
8953		Maximum recommended file transfer size.
8954		Minimum Acceptable Value: Not specified.
8955	ADV	{POSIX_REC_MIN_XFER_SIZE}
8956		Minimum recommended file transfer size.
8957		Minimum Acceptable Value: Not specified.
8958	ADV	{POSIX_REC_XFER_ALIGN}
8959		Recommended file transfer buffer alignment.
8960		Minimum Acceptable Value: Not specified.
8961		{SYMLINK_MAX}
8962		Maximum number of bytes in a symbolic link.
8963		Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

#### 8964 Runtime Inceasable Values

8965 The magnitude limitations in the following list shall be fixed by specific implementations. An  
 8966 application should assume that the value of the symbolic constant defined by <limits.h> in a  
 8967 specific implementation is the minimum that pertains whenever the application is run under  
 8968 that implementation. A specific instance of a specific implementation may increase the value  
 8969 relative to that supplied by <limits.h> for that implementation. The actual value supported by a  
 8970 specific instance shall be provided by the *sysconf()* function.

8971	{BC_BASE_MAX}	
8972		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
8973		Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}
8974	{BC_DIM_MAX}	
8975		Maximum number of elements permitted in an array by the <i>bc</i> utility.
8976		Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}
8977	{BC_SCALE_MAX}	
8978		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
8979		Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}
8980	{BC_STRING_MAX}	
8981		Maximum length of a string constant accepted by the <i>bc</i> utility.
8982		Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}
8983	{CHARCLASS_NAME_MAX}	
8984		Maximum number of bytes in a character class name.
8985		Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}
8986	{COLL_WEIGHTS_MAX}	
8987		Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see <a href="#">Chapter 7</a> (on page 135).
8988		
8989		Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}
8990	{EXPR_NEST_MAX}	
8991		Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
8992		Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}
8993	{LINE_MAX}	
8994		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.
8995		
8996		Minimum Acceptable Value: {_POSIX2_LINE_MAX}
8997		
8998	{NGROUPS_MAX}	
8999		Maximum number of simultaneous supplementary group IDs per process.
9000		Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}
9001	{RE_DUP_MAX}	
9002		Maximum number of repeated occurrences of a regular expression permitted when using the interval notation $\{m,n\}$ ; see <a href="#">Chapter 9</a> (on page 181).
9003		
9004		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
9005		
9005	<b>Maximum Values</b>	
9006		The <limits.h> header shall define the following symbolic constants with the values shown.
9007		These are the most restrictive values for certain features on an implementation. A conforming implementation shall provide values no larger than these values. A conforming application must not require a smaller value for correct operation.
9008		
9009		
9010	{_POSIX_CLOCKRES_MIN}	
9011		The resolution of the CLOCK_REALTIME clock, in nanoseconds.
9012		Value: 20 000 000
9013	MON	If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
9014		clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

9015 **Minimum Values**

9016 The <limits.h> header shall define the following symbolic constants with the values shown.  
 9017 These are the most restrictive values for certain features on an implementation conforming to  
 9018 this volume of POSIX.1-2008. Related symbolic constants are defined elsewhere in this volume  
 9019 of POSIX.1-2008 which reflect the actual implementation and which need not be as restrictive. A  
 9020 conforming implementation shall provide values at least this large. A strictly conforming  
 9021 application must not require a larger value for correct operation.

9022 `{_POSIX_AIO_LISTIO_MAX}`

9023 The number of I/O operations that can be specified in a list I/O call.

9024 Value: 2

9025 `{_POSIX_AIO_MAX}`

9026 The number of outstanding asynchronous I/O operations.

9027 Value: 1

9028 `{_POSIX_ARG_MAX}`

9029 Maximum length of argument to the *exec* functions including environment data.

9030 Value: 4 096

9031 `{_POSIX_CHILD_MAX}`

9032 Maximum number of simultaneous processes per real user ID.

9033 Value: 25

9034 `{_POSIX_DELAYTIMER_MAX}`

9035 The number of timer expiration overruns.

9036 Value: 32

9037 `{_POSIX_HOST_NAME_MAX}`

9038 Maximum length of a host name (not including the terminating null) as returned from the  
 9039 *gethostname()* function.

9040 Value: 255

9041 `{_POSIX_LINK_MAX}`

9042 Maximum number of links to a single file.

9043 Value: 8

9044 `{_POSIX_LOGIN_NAME_MAX}`

9045 The size of the storage required for a login name, in bytes (including the terminating null).

9046 Value: 9

9047 `{_POSIX_MAX_CANON}`

9048 Maximum number of bytes in a terminal canonical input queue.

9049 Value: 255

9050 `{_POSIX_MAX_INPUT}`

9051 Maximum number of bytes allowed in a terminal input queue.

9052 Value: 255

9053 MSG `{_POSIX_MQ_OPEN_MAX}`

9054 The number of message queues that can be open for a single process.

9055 Value: 8

9056 MSG `{_POSIX_MQ_PRIO_MAX}`

9057 The maximum number of message priorities supported by the implementation.

9058 Value: 32

## Headers

## &lt;limits.h&gt;

9059	{_POSIX_NAME_MAX}	
9060		Maximum number of bytes in a filename (not including the terminating null).
9061		Value: 14
9062	{_POSIX_NGROUPS_MAX}	
9063		Maximum number of simultaneous supplementary group IDs per process.
9064		Value: 8
9065	{_POSIX_OPEN_MAX}	
9066		Maximum number of files that one process can have open at any one time.
9067		Value: 20
9068	{_POSIX_PATH_MAX}	
9069		Minimum number the implementation will accept as the maximum number of bytes in a
9070		pathname.
9071		Value: 256
9072	{_POSIX_PIPE_BUF}	
9073		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9074		Value: 512
9075	{_POSIX_RE_DUP_MAX}	
9076		The number of repeated occurrences of a BRE permitted by the <i>regex</i> ( <i>re</i> ) and <i>regcomp</i> ( <i>re</i> )
9077		functions when using the interval notation $\{m,n\}$ ; see Section 9.3.6 (on page 186).
9078		Value: 255
9079	{_POSIX_RTSIG_MAX}	
9080		The number of realtime signal numbers reserved for application use.
9081		Value: 8
9082	{_POSIX_SEM_NSEMS_MAX}	
9083		The number of semaphores that a process may have.
9084		Value: 256
9085	{_POSIX_SEM_VALUE_MAX}	
9086		The maximum value a semaphore may have.
9087		Value: 32 767
9088	{_POSIX_SIGQUEUE_MAX}	
9089		The number of queued signals that a process may send and have pending at the receiver(s)
9090		at any time.
9091		Value: 32
9092	{_POSIX_SSIZE_MAX}	
9093		The value that can be stored in an object of type <i>ssize_t</i> .
9094		Value: 32 767
9095	SS TSP {_POSIX_SS_REPL_MAX}	
9096		The number of replenishment operations that may be simultaneously pending for a
9097		particular sporadic server scheduler.
9098		Value: 4
9099	{_POSIX_STREAM_MAX}	
9100		The number of streams that one process can have open at one time.
9101		Value: 8

9102		{_POSIX_SYMLINK_MAX}
9103		The number of bytes in a symbolic link.
9104		Value: 255
9105		{_POSIX_SYMLOOP_MAX}
9106		The number of symbolic links that can be traversed in the resolution of a pathname in the absence of a loop.
9107		Value: 8
9108		
9109		{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9110		The number of attempts made to destroy a thread's thread-specific data values on thread exit.
9111		Value: 4
9112		
9113		{_POSIX_THREAD_KEYS_MAX}
9114		The number of data keys per process.
9115		Value: 128
9116		{_POSIX_THREAD_THREADS_MAX}
9117		The number of threads per process.
9118		Value: 64
9119		{_POSIX_TIMER_MAX}
9120		The per-process number of timers.
9121		Value: 32
9122	OB TRC	{_POSIX_TRACE_EVENT_NAME_MAX}
9123		The length in bytes of a trace event name (not including the terminating null).
9124		Value: 30
9125	OB TRC	{_POSIX_TRACE_NAME_MAX}
9126		The length in bytes of a trace generation version string or a trace stream name (not including the terminating null).
9127		Value: 8
9128		
9129	OB TRC	{_POSIX_TRACE_SYS_MAX}
9130		The number of trace streams that may simultaneously exist in the system.
9131		Value: 8
9132	OB TRC	{_POSIX_TRACE_USER_EVENT_MAX}
9133		The number of user trace event type identifiers that may simultaneously exist in a traced process, including the predefined user trace event
9134		POSIX_TRACE_UNNAMED_USER_EVENT.
9135		Value: 32
9136		
9137		{_POSIX_TTY_NAME_MAX}
9138		The size of the storage required for a terminal device name, in bytes (including the terminating null).
9139		Value: 9
9140		
9141		{_POSIX_TZNAME_MAX}
9142		Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
9143		Value: 6
9144		<b>Note:</b> The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters mentioned in <a href="#">Section 8.3</a> (on page 177).
9145		

## Headers

## &lt;limits.h&gt;

9146		<code>{_POSIX2_BC_BASE_MAX}</code>
9147		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
9148		Value: 99
9149		<code>{_POSIX2_BC_DIM_MAX}</code>
9150		Maximum number of elements permitted in an array by the <i>bc</i> utility.
9151		Value: 2 048
9152		<code>{_POSIX2_BC_SCALE_MAX}</code>
9153		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
9154		Value: 99
9155		<code>{_POSIX2_BC_STRING_MAX}</code>
9156		Maximum length of a string constant accepted by the <i>bc</i> utility.
9157		Value: 1 000
9158		<code>{_POSIX2_CHARCLASS_NAME_MAX}</code>
9159		Maximum number of bytes in a character class name.
9160		Value: 14
9161		<code>{_POSIX2_COLL_WEIGHTS_MAX}</code>
9162		Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see <a href="#">Chapter 7</a> (on page 135).
9163		Value: 2
9164		
9165		<code>{_POSIX2_EXPR_NEST_MAX}</code>
9166		Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
9167		Value: 32
9168		<code>{_POSIX2_LINE_MAX}</code>
9169		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.
9170		Value: 2 048
9171		
9172		
9173		<code>{_POSIX2_RE_DUP_MAX}</code>
9174		Maximum number of repeated occurrences of a regular expression permitted when using the interval notation <code>\{m,n\}</code> ; see <a href="#">Chapter 9</a> (on page 181).
9175		Value: 255
9176		
9177	XSI	<code>{_XOPEN_IOV_MAX}</code>
9178		Maximum number of <b>iovec</b> structures that one process has available for use with <i>readv()</i> or <i>writev()</i> .
9179		Value: 16
9180		
9181	XSI	<code>{_XOPEN_NAME_MAX}</code>
9182		Maximum number of bytes in a filename (not including the terminating null).
9183		Value: 255
9184	XSI	<code>{_XOPEN_PATH_MAX}</code>
9185		Minimum number the implementation will accept as the maximum number of bytes in a pathname.
9186		Value: 1 024
9187		

## 9188 Numerical Limits

9189 The <limits.h> header shall define the following macros and, except for {CHAR\_BIT},  
 9190 {LONG\_BIT}, {MB\_LEN\_MAX}, and {WORD\_BIT}, they shall be replaced by expressions that  
 9191 have the same type as would an expression that is an object of the corresponding type converted  
 9192 according to the integer promotions.

9193 If the value of an object of type **char** is treated as a signed integer when used in an expression,  
 9194 the value of {CHAR\_MIN} is the same as that of {SCHAR\_MIN} and the value of {CHAR\_MAX}  
 9195 is the same as that of {SCHAR\_MAX}. Otherwise, the value of {CHAR\_MIN} is 0 and the value  
 9196 of {CHAR\_MAX} is the same as that of {UCHAR\_MAX}.

9197 {CHAR\_BIT}

9198 Number of bits in a type **char**.

9199 CX Value: 8

9200 {CHAR\_MAX}

9201 Maximum value for an object of type **char**.

9202 Value: {UCHAR\_MAX} or {SCHAR\_MAX}

9203 {CHAR\_MIN}

9204 Minimum value for an object of type **char**.

9205 Value: {SCHAR\_MIN} or 0

9206 {INT\_MAX}

9207 Maximum value for an object of type **int**.

9208 CX Minimum Acceptable Value: 2 147 483 647

9209 {INT\_MIN}

9210 Minimum value for an object of type **int**.

9211 CX Maximum Acceptable Value: -2 147 483 647

9212 {LLONG\_MAX}

9213 Maximum value for an object of type **long long**.

9214 Minimum Acceptable Value: +9 223 372 036 854 775 807

9215 {LLONG\_MIN}

9216 Minimum value for an object of type **long long**.

9217 Maximum Acceptable Value: -9 223 372 036 854 775 807

9218 {LONG\_BIT}

9219 Number of bits in an object of type **long**.

9220 Minimum Acceptable Value: 32

9221 {LONG\_MAX}

9222 Maximum value for an object of type **long**.

9223 Minimum Acceptable Value: +2 147 483 647

9224 {LONG\_MIN}

9225 Minimum value for an object of type **long**.

9226 Maximum Acceptable Value: -2 147 483 647

9227 {MB\_LEN\_MAX}

9228 Maximum number of bytes in a character, for any supported locale.

9229 Minimum Acceptable Value: 1

9230 {SCHAR\_MAX}

9231 Maximum value for an object of type **signed char**.

9232	CX	Value: +127
9233		{SCHAR_MIN}
9234		Minimum value for an object of type <b>signed char</b> .
9235	CX	Value: -128
9236		{SHRT_MAX}
9237		Maximum value for an object of type <b>short</b> .
9238		Minimum Acceptable Value: +32 767
9239		{SHRT_MIN}
9240		Minimum value for an object of type <b>short</b> .
9241		Maximum Acceptable Value: -32 767
9242		{SSIZE_MAX}
9243		Maximum value for an object of type <b>ssize_t</b> .
9244		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}
9245		{UCHAR_MAX}
9246		Maximum value for an object of type <b>unsigned char</b> .
9247	CX	Value: 255
9248		{UINT_MAX}
9249		Maximum value for an object of type <b>unsigned</b> .
9250	CX	Minimum Acceptable Value: 4 294 967 295
9251		{ULLONG_MAX}
9252		Maximum value for an object of type <b>unsigned long long</b> .
9253		Minimum Acceptable Value: 18 446 744 073 709 551 615
9254		{ULONG_MAX}
9255		Maximum value for an object of type <b>unsigned long</b> .
9256		Minimum Acceptable Value: 4 294 967 295
9257		{USHRT_MAX}
9258		Maximum value for an object of type <b>unsigned short</b> .
9259		Minimum Acceptable Value: 65 535
9260		{WORD_BIT}
9261		Number of bits in an object of type <b>int</b> .
9262		Minimum Acceptable Value: 32
9263		<b>Other Invariant Values</b>
9264		The <limits.h> header shall define the following symbolic constants:
9265		{NL_ARGMAX}
9266		Maximum value of <i>n</i> in conversion specifications using the "%n\$" sequence in calls to the <i>printf()</i> and <i>scanf()</i> families of functions.
9267		Minimum Acceptable Value: 9
9269	XSI	{NL_LANGMAX}
9270		Maximum number of bytes in a <i>LANG</i> name.
9271		Minimum Acceptable Value: 14
9272		{NL_MSGMAX}
9273		Maximum message number.
9274		Minimum Acceptable Value: 32 767

9275 {NL\_SETMAX}  
 9276 Maximum set number.  
 9277 Minimum Acceptable Value: 255

9278 {NL\_TEXTMAX}  
 9279 Maximum number of bytes in a message string.  
 9280 Minimum Acceptable Value: {\_POSIX2\_LINE\_MAX}

9281 XSI {NZERO}  
 9282 Default process priority.  
 9283 Minimum Acceptable Value: 20

**9284 APPLICATION USAGE**

9285 None.

**9286 RATIONALE**

9287 A request was made to reduce the value of {\_POSIX\_LINK\_MAX} from the value of 8 specified  
 9288 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request  
 9289 for several reasons:

- 9290 • They wanted to avoid making any changes to the standard that could break conforming  
 9291 applications, and the requested change could have that effect.
- 9292 • The use of multiple hard links to a file cannot always be replaced with use of symbolic  
 9293 links. Symbolic links are semantically different from hard links in that they associate a  
 9294 pathname with another pathname rather than a pathname with a file. This has  
 9295 implications for access control, file permanence, and transparency.
- 9296 • The original standard developers had considered the issue of allowing for  
 9297 implementations that did not in general support hard links, and decided that this would  
 9298 reduce consensus on the standard.

9299 Systems that support historical versions of the development option of the ISO POSIX-2 standard  
 9300 retain the name {\_POSIX2\_RE\_DUP\_MAX} as an alias for {\_POSIX\_RE\_DUP\_MAX}.

9301 {PATH\_MAX}  
 9302 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the  
 9303 definition of pathname and the description of {PATH\_MAX}, allowing application  
 9304 developers to allocate either {PATH\_MAX} or {PATH\_MAX}+1 bytes. The inconsistency has  
 9305 been removed by correction to the {PATH\_MAX} definition to include the null character.  
 9306 With this change, applications that previously allocated {PATH\_MAX} bytes will continue to  
 9307 succeed.

9308 {SYMLINK\_MAX}  
 9309 This symbol refers to space for data that is stored in the file system, as opposed to  
 9310 {PATH\_MAX} which is the length of a name that can be passed to a function. In some  
 9311 existing implementations, the filenames pointed to by symbolic links are stored in the *inodes*  
 9312 of the links, so it is important that {SYMLINK\_MAX} not be constrained to be as large as  
 9313 {PATH\_MAX}.

**9314 FUTURE DIRECTIONS**

9315 None.

**9316 SEE ALSO**

9317 Chapter 7 (on page 135), <stdio.h>, <unistd.h>  
 9318 XSH Section 2.2 (on page 468), *fpathconf()*, *sysconf()*

9319 **CHANGE HISTORY**

9320 First released in Issue 1.

9321 **Issue 5**9322 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
9323 Threads Extension.

9324 {FILESIZEBITS} is added for the Large File Summit extensions.

9325 The minimum acceptable values for {INT\_MAX}, {INT\_MIN}, and {UINT\_MAX} are changed to  
9326 make 32-bit values the minimum requirement.

9327 The entry is restructured to improve readability.

9328 **Issue 6**9329 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR\_MIN},  
9330 {INT\_MIN}, {LONG\_MIN}, {SCHAR\_MIN}, and {SHRT\_MIN} that these are maximum  
9331 acceptable values.9332 The following new requirements on POSIX implementations derive from alignment with the  
9333 Single UNIX Specification:

- 9334 • The minimum value for {CHILD\_MAX} is 25. This is a FIPS requirement.
- 9335 • The minimum value for {OPEN\_MAX} is 20. This is a FIPS requirement.
- 9336 • The minimum value for {NGROUPS\_MAX} is 8. This is also a FIPS requirement.

9337 Symbolic constants are added for {\_POSIX\_SYMLINK\_MAX}, {\_POSIX\_SYMLOOP\_MAX},  
9338 {\_POSIX\_RE\_DUP\_MAX}, {RE\_DUP\_MAX}, {SYMLOOP\_MAX}, and {SYMLINK\_MAX}.

9339 The following values are added for alignment with IEEE Std 1003.1d-1999:

9340 {\_POSIX\_SS\_REPL\_MAX}  
 9341 {SS\_REPL\_MAX}  
 9342 {POSIX\_ALLOC\_SIZE\_MIN}  
 9343 {POSIX\_REC\_INCR\_XFER\_SIZE}  
 9344 {POSIX\_REC\_MAX\_XFER\_SIZE}  
 9345 {POSIX\_REC\_MIN\_XFER\_SIZE}  
 9346 {POSIX\_REC\_XFER\_ALIGN}

9347 Reference to CLOCK\_MONOTONIC is added in the description of {\_POSIX\_CLOCKRES\_MIN}  
9348 for alignment with IEEE Std 1003.1j-2000.9349 The constants {LLONG\_MIN}, {LLONG\_MAX}, and {ULLONG\_MAX} are added for alignment  
9350 with the ISO/IEC 9899:1999 standard.

9351 The following values are added for alignment with IEEE Std 1003.1q-2000:

9352 {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}  
 9353 {\_POSIX\_TRACE\_NAME\_MAX}  
 9354 {\_POSIX\_TRACE\_SYS\_MAX}  
 9355 {\_POSIX\_TRACE\_USER\_EVENT\_MAX}  
 9356 {TRACE\_EVENT\_NAME\_MAX}  
 9357 {TRACE\_NAME\_MAX}  
 9358 {TRACE\_SYS\_MAX}  
 9359 {TRACE\_USER\_EVENT\_MAX}

9360 The new limits {\_XOPEN\_NAME\_MAX} and {\_XOPEN\_PATH\_MAX} are added as minimum

- 9361 values for {PATH\_MAX} and {NAME\_MAX} limits on XSI-conformant systems.
- 9362 The LEGACY symbols {PASS\_MAX} and {TMP\_MAX} are removed.
- 9363 The values for the limits {CHAR\_BIT}, {SCHAR\_MAX}, and {UCHAR\_MAX} are now required  
9364 to be 8, +127, and 255, respectively.
- 9365 The value for the limit {CHAR\_MAX} is now {UCHAR\_MAX} or {SCHAR\_MAX}.
- 9366 The value for the limit {CHAR\_MIN} is now {SCHAR\_MIN} or zero.
- 9367 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of  
9368 {\_POSIX\_CHILD\_MAX} from 6 to 25. This is for FIPS 151-2 alignment.
- 9369 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for  
9370 {INT\_MAX}, {UINT\_MAX}, and {INT\_MIN} to be CX extensions over the ISO C standard, and  
9371 correcting {WORD\_BIT} from 16 to 32.
- 9372 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing  
9373 {CHARCLASS\_NAME\_MAX} from the “Other Invariant Values” section (it also occurs under  
9374 “Runtime Inceasable Values”).
- 9375 **Issue 7**
- 9376 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.
- 9377 Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of  
9378 {TRACE\_EVENT\_NAME\_MAX} and {TRACE\_NAME\_MAX} to not include the terminating  
9379 null.
- 9380 SD5-XBD-ERN-36 is applied, changing the description of {RE\_DUP\_MAX}.
- 9381 SD5-XBD-ERN-90 is applied.
- 9382 {NL\_NMAX} is removed; it should have been removed in Issue 6.
- 9383 The Trace option values are marked obsolescent.
- 9384 The {ATEXIT\_MAX}, {LONG\_BIT}, {NL\_MSGMAX}, {NL\_SETMAX}, {NL\_TEXTMAX}, and  
9385 {WORD\_BIT} values are moved from the XSI option to the Base.
- 9386 The AIO\_\* and \_POSIX\_AIO\_\* values are moved from the Asynchronous Input and Output  
9387 option to the Base.
- 9388 The {\_POSIX\_RTSIG\_MAX}, {\_POSIX\_SIGQUEUE\_MAX}, {RTSIG\_MAX}, and  
9389 {SIGQUEUE\_MAX} values are moved from the Realtime Signals Extension option to the Base.
- 9390 Functionality relating to the Threads and Timers options is moved to the Base.
- 9391 This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;locale.h&gt;

## 9392 NAME

9393 locale.h — category macros

## 9394 SYNOPSIS

9395 #include &lt;locale.h&gt;

## 9396 DESCRIPTION

9397 CX Some of the functionality described on this reference page extends the ISO C standard.  
 9398 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 9399 enable the visibility of these symbols in this header.

9400 The <locale.h> header shall define the **lconv** structure, which shall include at least the following  
 9401 members. (See the definitions of *LC\_MONETARY* in Section 7.3.3 (on page 154) and Section 7.3.4  
 9402 (on page 157).)

9403 char \*currency\_symbol  
 9404 char \*decimal\_point  
 9405 char frac\_digits  
 9406 char \*grouping  
 9407 char \*int\_curr\_symbol  
 9408 char int\_frac\_digits  
 9409 char int\_n\_cs\_precedes  
 9410 char int\_n\_sep\_by\_space  
 9411 char int\_n\_sign\_posn  
 9412 char int\_p\_cs\_precedes  
 9413 char int\_p\_sep\_by\_space  
 9414 char int\_p\_sign\_posn  
 9415 char \*mon\_decimal\_point  
 9416 char \*mon\_grouping  
 9417 char \*mon\_thousands\_sep  
 9418 char \*negative\_sign  
 9419 char n\_cs\_precedes  
 9420 char n\_sep\_by\_space  
 9421 char n\_sign\_posn  
 9422 char \*positive\_sign  
 9423 char p\_cs\_precedes  
 9424 char p\_sep\_by\_space  
 9425 char p\_sign\_posn  
 9426 char \*thousands\_sep

9427 The <locale.h> header shall define NULL (as described in <stddef.h>) and at least the following  
 9428 as macros:

9429 *LC\_ALL*9430 *LC\_COLLATE*9431 *LC\_CTYPE*9432 CX *LC\_MESSAGES*9433 *LC\_MONETARY*9434 *LC\_NUMERIC*9435 *LC\_TIME*

9436 which shall expand to integer constant expressions with distinct values for use as the first  
 9437 argument to the *setlocale()* function.

9438 Implementations may add additional masks using the form *LC\_\** and an uppercase letter.

9439 CX The <locale.h> header shall contain at least the following macros representing bitmasks for use  
 9440 with the *newlocale()* function for each supported locale category:

9441 LC\_COLLATE\_MASK  
 9442 LC\_CTYPE\_MASK  
 9443 LC\_MESSAGES\_MASK  
 9444 LC\_MONETARY\_MASK  
 9445 LC\_NUMERIC\_MASK  
 9446 LC\_TIME\_MASK

9447 Implementations may add additional masks using the form LC\_\*\_MASK.

9448 In addition, a macro to set the bits for all categories set shall be defined:

9449 LC\_ALL\_MASK

9450 The <locale.h> header shall define LC\_GLOBAL\_LOCALE, a special locale object descriptor  
 9451 used by the *uselocale()* function.

9452 The <locale.h> header shall define the **locale\_t** type, representing a locale object.

9453 The following shall be declared as functions and may also be defined as macros. Function  
 9454 prototypes shall be provided for use with ISO C standard compilers.

9455 CX locale\_t duplocale(locale\_t);  
 9456 void freelocale(locale\_t);  
 9457 struct lconv \*localeconv(void);

9458 CX locale\_t newlocale(int, const char \*, locale\_t);  
 9459 char \*setlocale(int, const char \*);

9460 CX locale\_t uselocale(locale\_t);

**9461 APPLICATION USAGE**

9462 None.

**9463 RATIONALE**

9464 None.

**9465 FUTURE DIRECTIONS**

9466 None.

**9467 SEE ALSO**9468 [Chapter 8](#) (on page 173), [<stddef.h>](#)9469 XSH [duplocale\(\)](#), [freelocale\(\)](#), [localeconv\(\)](#), [newlocale\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)**9470 CHANGE HISTORY**

9471 First released in Issue 3.

9472 Included for alignment with the ISO C standard.

**9473 Issue 6**9474 The **lconv** structure is expanded with new members (**int\_n\_cs\_precedes**, **int\_n\_sep\_by\_space**,  
 9475 **int\_n\_sign\_posn**, **int\_p\_cs\_precedes**, **int\_p\_sep\_by\_space**, and **int\_p\_sign\_posn**) for alignment  
 9476 with the ISO/IEC 9899:1999 standard.

9477 Extensions beyond the ISO C standard are marked.

*Headers***<locale.h>**9478 **Issue 7**

9479 The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open  
9480 Group Technical Standard, 2006, Extended API Set Part 4.

9481 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<math.h>**9482 **NAME**9483 `math.h` — mathematical declarations9484 **SYNOPSIS**9485 `#include <math.h>`9486 **DESCRIPTION**

9487 CX Some of the functionality described on this reference page extends the ISO C standard.  
 9488 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 9489 enable the visibility of these symbols in this header.

9490 The **<math.h>** header shall define at least the following types:9491 **float\_t** A real-floating type at least as wide as **float**.9492 **double\_t** A real-floating type at least as wide as **double**, and at least as wide as **float\_t**.

9493 If FLT\_EVAL\_METHOD equals 0, **float\_t** and **double\_t** shall be **float** and **double**, respectively; if  
 9494 FLT\_EVAL\_METHOD equals 1, they shall both be **double**; if FLT\_EVAL\_METHOD equals 2,  
 9495 they shall both be **long double**; for other values of FLT\_EVAL\_METHOD, they are otherwise  
 9496 implementation-defined.

9497 The **<math.h>** header shall define the following macros, where real-floating indicates that the  
 9498 argument shall be an expression of real-floating type:

```
9499 int fpclassify(real-floating x);
9500 int isfinite(real-floating x);
9501 int isgreater(real-floating x, real-floating y);
9502 int isgreaterequal(real-floating x, real-floating y);
9503 int isinf(real-floating x);
9504 int isless(real-floating x, real-floating y);
9505 int islessequal(real-floating x, real-floating y);
9506 int islessgreater(real-floating x, real-floating y);
9507 int isnan(real-floating x);
9508 int isnormal(real-floating x);
9509 int isunordered(real-floating x, real-floating y);
9510 int signbit(real-floating x);
```

9511 The **<math.h>** header shall define the following symbolic constants. The values shall have type  
 9512 **double** and shall be accurate within the precision of the **double** type.

9513	XSI	<b>M_E</b>	Value of $e$
9514	XSI	<b>M_LOG2E</b>	Value of $\log_2 e$
9515	XSI	<b>M_LOG10E</b>	Value of $\log_{10} e$
9516	XSI	<b>M_LN2</b>	Value of $\log_e 2$
9517	XSI	<b>M_LN10</b>	Value of $\log_e 10$
9518	XSI	<b>M_PI</b>	Value of $\pi$
9519	XSI	<b>M_PI_2</b>	Value of $\pi/2$
9520	XSI	<b>M_PI_4</b>	Value of $\pi/4$
9521	XSI	<b>M_1_PI</b>	Value of $1/\pi$
9522	XSI	<b>M_2_PI</b>	Value of $2/\pi$

## Headers

## &lt;math.h&gt;

9523 XSI `M_2_SQRTPI` Value of  $2/\sqrt{\pi}$

9524 XSI `M_SQRT2` Value of  $\sqrt{2}$

9525 XSI `M_SQRT1_2` Value of  $1/\sqrt{2}$

9526 The <math.h> header shall define the following symbolic constant:

9527 OB XSI `MAXFLOAT` Same value as `FLT_MAX` in <float.h>.

9528 The <math.h> header shall define the following macros:

9529 `HUGE_VAL` A positive **double** constant expression, not necessarily representable as a  
9530 **float**. Used as an error value returned by the mathematics library.  
9531 `HUGE_VAL` evaluates to +infinity on systems supporting IEEE Std 754-1985.

9532 `HUGE_VALF` A positive **float** constant expression. Used as an error value returned by the  
9533 mathematics library. `HUGE_VALF` evaluates to +infinity on systems  
9534 supporting IEEE Std 754-1985.

9535 `HUGE_VALL` A positive **long double** constant expression. Used as an error value returned  
9536 by the mathematics library. `HUGE_VALL` evaluates to +infinity on systems  
9537 supporting IEEE Std 754-1985.

9538 `INFINITY` A constant expression of type **float** representing positive or unsigned infinity,  
9539 if available; else a positive constant of type **float** that overflows at translation  
9540 time.

9541 `NAN` A constant expression of type **float** representing a quiet NaN. This macro is  
9542 only defined if the implementation supports quiet NaNs for the **float** type.

9543 The following macros shall be defined for number classification. They represent the mutually-  
9544 exclusive kinds of floating-point values. They expand to integer constant expressions with  
9545 distinct values. Additional implementation-defined floating-point classifications, with macro  
9546 definitions beginning with `FP_` and an uppercase letter, may also be specified by the  
9547 implementation.

9548 `FP_INFINITE`  
9549 `FP_NAN`  
9550 `FP_NORMAL`  
9551 `FP_SUBNORMAL`  
9552 `FP_ZERO`

9553 The following optional macros indicate whether the `fma()` family of functions are fast compared  
9554 with direct code:

9555 `FP_FAST_FMA`  
9556 `FP_FAST_FMAF`  
9557 `FP_FAST_FMAL`

9558 If defined, the `FP_FAST_FMA` macro shall expand to the integer constant 1 and shall indicate  
9559 that the `fma()` function generally executes about as fast as, or faster than, a multiply and an add  
9560 of **double** operands. If undefined, the speed of execution is unspecified. The other macros have  
9561 the equivalent meaning for the **float** and **long double** versions.

9562 The following macros shall expand to integer constant expressions whose values are returned by  
9563 `ilogb(x)` if  $x$  is zero or NaN, respectively. The value of `FP_ILOGB0` shall be either `{INT_MIN}` or  
9564 `-{INT_MAX}`. The value of `FP_ILOGBNAN` shall be either `{INT_MAX}` or `{INT_MIN}`.

9565 FP\_ILOGB0  
 9566 FP\_ILOGBNAN

9567 The following macros shall expand to the integer constants 1 and 2, respectively;

9568 MATH\_ERRNO  
 9569 MATH\_ERREXCEPT

9570 The following macro shall expand to an expression that has type **int** and the value  
 9571 MATH\_ERRNO, MATH\_ERREXCEPT, or the bitwise-inclusive OR of both:

9572 math\_errhandling

9573 The value of math\_errhandling is constant for the duration of the program. It is unspecified  
 9574 whether math\_errhandling is a macro or an identifier with external linkage. If a macro definition  
 9575 is suppressed or a program defines an identifier with the name math\_errhandling, the behavior  
 9576 is undefined. If the expression (math\_errhandling & MATH\_ERREXCEPT) can be non-zero, the  
 9577 implementation shall define the macros FE\_DIVBYZERO, FE\_INVALID, and FE\_OVERFLOW in  
 9578 <fenv.h>.

9579 The following shall be declared as functions and may also be defined as macros. Function  
 9580 prototypes shall be provided.

9581 double acos(double);  
 9582 float acosf(float);  
 9583 double acosh(double);  
 9584 float acoshf(float);  
 9585 long double acoshl(long double);  
 9586 long double acosl(long double);  
 9587 double asin(double);  
 9588 float asinf(float);  
 9589 double asinh(double);  
 9590 float asinhf(float);  
 9591 long double asinhl(long double);  
 9592 long double asinl(long double);  
 9593 double atan(double);  
 9594 double atan2(double, double);  
 9595 float atan2f(float, float);  
 9596 long double atan2l(long double, long double);  
 9597 float atanf(float);  
 9598 double atanh(double);  
 9599 float atanhf(float);  
 9600 long double atanh1(long double);  
 9601 long double atanl(long double);  
 9602 double cbrt(double);  
 9603 float cbrtf(float);  
 9604 long double cbrtl(long double);  
 9605 double ceil(double);  
 9606 float ceilf(float);  
 9607 long double ceill(long double);  
 9608 double copysign(double, double);  
 9609 float copysignf(float, float);  
 9610 long double copysignl(long double, long double);  
 9611 double cos(double);

## Headers

## &lt;math.h&gt;

9612 float cosf(float);  
 9613 double cosh(double);  
 9614 float coshf(float);  
 9615 long double coshl(long double);  
 9616 long double cosl(long double);  
 9617 double erf(double);  
 9618 double erfc(double);  
 9619 float erfcf(float);  
 9620 long double erfcl(long double);  
 9621 float erff(float);  
 9622 long double erfl(long double);  
 9623 double exp(double);  
 9624 double exp2(double);  
 9625 float exp2f(float);  
 9626 long double exp2l(long double);  
 9627 float expf(float);  
 9628 long double expl(long double);  
 9629 double expm1(double);  
 9630 float expm1f(float);  
 9631 long double expm1l(long double);  
 9632 double fabs(double);  
 9633 float fabsf(float);  
 9634 long double fabsl(long double);  
 9635 double fdim(double, double);  
 9636 float fdimf(float, float);  
 9637 long double fdiml(long double, long double);  
 9638 double floor(double);  
 9639 float floorf(float);  
 9640 long double floorl(long double);  
 9641 double fma(double, double, double);  
 9642 float fmaf(float, float, float);  
 9643 long double fmal(long double, long double, long double);  
 9644 double fmax(double, double);  
 9645 float fmaxf(float, float);  
 9646 long double fmaxl(long double, long double);  
 9647 double fmin(double, double);  
 9648 float fminf(float, float);  
 9649 long double fminl(long double, long double);  
 9650 double fmod(double, double);  
 9651 float fmodf(float, float);  
 9652 long double fmodl(long double, long double);  
 9653 double frexp(double, int \*);  
 9654 float frexpf(float, int \*);  
 9655 long double frexpl(long double, int \*);  
 9656 double hypot(double, double);  
 9657 float hypotf(float, float);  
 9658 long double hypotl(long double, long double);  
 9659 int ilogb(double);  
 9660 int ilogbf(float);  
 9661 int ilogbl(long double);  
 9662 XSI double j0(double);  
 9663 double j1(double);

```

9664     double      jn(int, double);
9665     double      ldexp(double, int);
9666     float       ldexpf(float, int);
9667     long double  ldexpl(long double, int);
9668     double      lgamma(double);
9669     float       lgammaf(float);
9670     long double  lgammal(long double);
9671     long long    llrint(double);
9672     long long    llrintf(float);
9673     long long    llrintl(long double);
9674     long long    llround(double);
9675     long long    llroundf(float);
9676     long long    llroundl(long double);
9677     double      log(double);
9678     double      log10(double);
9679     float       log10f(float);
9680     long double  log10l(long double);
9681     double      loglp(double);
9682     float       loglpf(float);
9683     long double  loglpl(long double);
9684     double      log2(double);
9685     float       log2f(float);
9686     long double  log2l(long double);
9687     double      logb(double);
9688     float       logbf(float);
9689     long double  logbl(long double);
9690     float       logf(float);
9691     long double  logl(long double);
9692     long        lrint(double);
9693     long        lrintf(float);
9694     long        lrintl(long double);
9695     long        lround(double);
9696     long        lroundf(float);
9697     long        lroundl(long double);
9698     double      modf(double, double *);
9699     float       modff(float, float *);
9700     long double  modfl(long double, long double *);
9701     double      nan(const char *);
9702     float       nanf(const char *);
9703     long double  nanl(const char *);
9704     double      nearbyint(double);
9705     float       nearbyintf(float);
9706     long double  nearbyintl(long double);
9707     double      nextafter(double, double);
9708     float       nextafterf(float, float);
9709     long double  nextafterl(long double, long double);
9710     double      nexttoward(double, long double);
9711     float       nexttowardf(float, long double);
9712     long double  nexttowardl(long double, long double);
9713     double      pow(double, double);
9714     float       powf(float, float);
9715     long double  powl(long double, long double);

```

## Headers

## &lt;math.h&gt;

```

9716     double    remainder(double, double);
9717     float      remainderf(float, float);
9718     long double remainderl(long double, long double);
9719     double     remquo(double, double, int *);
9720     float      remquof(float, float, int *);
9721     long double remquol(long double, long double, int *);
9722     double     rint(double);
9723     float      rintf(float);
9724     long double rintl(long double);
9725     double     round(double);
9726     float      roundf(float);
9727     long double roundl(long double);
9728     double     scalbn(double, long);
9729     float      scalblnf(float, long);
9730     long double scalblnl(long double, long);
9731     double     scalbn(double, int);
9732     float      scalbnf(float, int);
9733     long double scalbnl(long double, int);
9734     double     sin(double);
9735     float      sinf(float);
9736     double     sinh(double);
9737     float      sinhf(float);
9738     long double sinhl(long double);
9739     long double sinl(long double);
9740     double     sqrt(double);
9741     float      sqrtf(float);
9742     long double sqrtl(long double);
9743     double     tan(double);
9744     float      tanf(float);
9745     double     tanh(double);
9746     float      tanhf(float);
9747     long double tanhl(long double);
9748     long double tanl(long double);
9749     double     tgamma(double);
9750     float      tgammaf(float);
9751     long double tgamma_l(long double);
9752     double     trunc(double);
9753     float      truncf(float);
9754     long double trunc_l(long double);
9755     double     y0(double);
9756     double     y1(double);
9757     double     yn(int, double);

```

9758 The following external variable shall be defined:

```

9759     extern int signgam;

```

9760 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces  
9761 volume of POSIX.1-2008 for all representable values of its input arguments, except where stated  
9762 otherwise. Each function shall execute as if it were a single operation without generating any  
9763 externally visible exceptional conditions.

9764 **APPLICATION USAGE**

9765 The FP\_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is  
 9766 off) the implementation to contract expressions. Each pragma can occur either outside external  
 9767 declarations or preceding all explicit declarations and statements inside a compound statement.  
 9768 When outside external declarations, the pragma takes effect from its occurrence until another  
 9769 FP\_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a  
 9770 compound statement, the pragma takes effect from its occurrence until another FP\_CONTRACT  
 9771 pragma is encountered (including within a nested compound statement), or until the end of the  
 9772 compound statement; at the end of a compound statement the state for the pragma is restored to  
 9773 its condition just before the compound statement. If this pragma is used in any other context, the  
 9774 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

9775 Applications should use FLT\_MAX as described in the <float.h> header instead of the  
 9776 obsolescent MAXFLOAT.

9777 **RATIONALE**

9778 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type  
 9779 **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved  
 9780 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999  
 9781 standard provides for all three versions of math functions.

9782 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their  
 9783 capability is available through *sprintf()*.

9784 **FUTURE DIRECTIONS**

9785 None.

9786 **SEE ALSO**

9787 <float.h>, <stddef.h>, <sys/types.h>

9788 XSH Section 2.2 (on page 468), *acos()*, *acosh()*, *asin()*, *asinh()*, *atan()*, *atan2()*, *atanh()*, *cbrt()*,  
 9789 *ceil()*, *copysign()*, *cos()*, *cosh()*, *erf()*, *erfc()*, *exp()*, *exp2()*, *expm1()*, *fabs()*, *fdim()*, *floor()*, *fma()*,  
 9790 *fmax()*, *fmin()*, *fmod()*, *fpclassify()*, *frexp()*, *hypot()*, *ilogb()*, *isfinite()*, *isgreater()*, *isgreaterequal()*,  
 9791 *isinf()*, *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *j0()*, *ldexp()*,  
 9792 *lgamma()*, *llrint()*, *llround()*, *log()*, *log10()*, *log1p()*, *log2()*, *logb()*, *lrint()*, *lround()*, *modf()*, *nan()*,  
 9793 *nearbyint()*, *nextafter()*, *pow()*, *remainder()*, *remquo()*, *rint()*, *round()*, *scalbln()*, *signbit()*, *sin()*,  
 9794 *sinh()*, *sqrt()*, *tan()*, *tanh()*, *tgamma()*, *trunc()*, *y0()*

9795 **CHANGE HISTORY**

9796 First released in Issue 1.

9797 **Issue 6**

9798 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

9799 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the  
 9800 meaning of the FP\_FAST\_FMA macro is unspecified if the macro is undefined.

9801 **Issue 7**

9802 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied,  
 9803 clarifying the wording of the FP\_FAST\_FMA macro.

9804 The MAXFLOAT constant is marked obsolescent.

9805 This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;monetary.h&gt;

9806 **NAME**

9807           monetary.h — monetary types

9808 **SYNOPSIS**

9809           #include &lt;monetary.h&gt;

9810 **DESCRIPTION**9811           The <monetary.h> header shall define the **locale\_t** type as described in <locale.h>.9812           The <monetary.h> header shall define the **size\_t** type as described in <stddef.h>.9813           The <monetary.h> header shall define the **ssize\_t** type as described in <sys/types.h>.9814           The following shall be declared as functions and may also be defined as macros. Function  
9815           prototypes shall be provided for use with ISO C standard compilers.

9816           ssize\_t strfmon(char \*restrict, size\_t, const char \*restrict, ...);

9817           ssize\_t strfmon\_l(char \*restrict, size\_t, locale\_t,

9818                       const char \*restrict, ...);

9819 **APPLICATION USAGE**

9820           None.

9821 **RATIONALE**

9822           None.

9823 **FUTURE DIRECTIONS**

9824           None.

9825 **SEE ALSO**

9826           &lt;locale.h&gt;, &lt;stddef.h&gt;, &lt;sys/types.h&gt;

9827           XSH *strfmon()*9828 **CHANGE HISTORY**

9829           First released in Issue 4.

9830 **Issue 6**9831           The **restrict** keyword is added to the prototype for *strfmon()*.9832 **Issue 7**

9833           The &lt;monetary.h&gt; header is moved from the XSI option to the Base.

9834           The *strmon\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
9835           Set Part 4.9836           A declaration for the **locale\_t** type is added.

**<mqueue.h>**9837 **NAME**9838 mqueue.h — message queues (**REALTIME**)9839 **SYNOPSIS**

9840 MSG #include &lt;mqueue.h&gt;

9841 **DESCRIPTION**9842 The **<mqueue.h>** header shall define the **mqd\_t** type, which is used for message queue  
9843 descriptors. This is not an array type.9844 The **<mqueue.h>** header shall define the **pthread\_attr\_t**, **size\_t**, and **ssize\_t** types as described  
9845 in **<sys/types.h>**.9846 The **<mqueue.h>** header shall define the **struct timespec** structure as described in **<time.h>**.9847 The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which  
9848 are described in the **<signal.h>** header.9849 The **<mqueue.h>** header shall define the **mq\_attr** structure, which is used in getting and setting  
9850 the attributes of a message queue. Attributes are initially set when the message queue is created.  
9851 An **mq\_attr** structure shall have at least the following fields:

9852	long	mq_flags	Message queue flags.
9853	long	mq_maxmsg	Maximum number of messages.
9854	long	mq_msgsize	Maximum message size.
9855	long	mq_curmsgs	Number of messages currently queued.

9856 The following shall be declared as functions and may also be defined as macros. Function  
9857 prototypes shall be provided.

```

9858 int      mq_close(mqd_t);
9859 int      mq_getattr(mqd_t, struct mq_attr *);
9860 int      mq_notify(mqd_t, const struct sigevent *);
9861 mqd_t    mq_open(const char *, int, ...);
9862 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
9863 int      mq_send(mqd_t, const char *, size_t, unsigned);
9864 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
9865                    struct mq_attr *restrict);
9866 ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
9867                        unsigned *restrict, const struct timespec *restrict);
9868 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
9869                     const struct timespec *);
9870 int      mq_unlink(const char *);

```

9871 Inclusion of the **<mqueue.h>** header may make visible symbols defined in the headers  
9872 **<fcntl.h>**, **<signal.h>**, and **<time.h>**.9873 **APPLICATION USAGE**

9874 None.

9875 **RATIONALE**

9876 None.

9877 **FUTURE DIRECTIONS**

9878 None.

9879 **SEE ALSO**9880 [<fcntl.h>](#), [<signal.h>](#), [<sys/types.h>](#), [<time.h>](#)9881 XSH [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#), [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#),  
9882 [mq\\_unlink\(\)](#)9883 **CHANGE HISTORY**

9884 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9885 **Issue 6**

9886 The &lt;mqqueue.h&gt; header is marked as part of the Message Passing option.

9887 The [mq\\_timedreceive\(\)](#) and [mq\\_timedsend\(\)](#) functions are added for alignment with IEEE Std  
9888 1003.1d-1999.9889 The **restrict** keyword is added to the prototypes for [mq\\_setattr\(\)](#) and [mq\\_timedreceive\(\)](#).9890 **Issue 7**

9891 Type and structure declarations are added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<ndbm.h>**

Headers

9892 **NAME**

9893 ndbm.h — definitions for ndbm database operations

9894 **SYNOPSIS**9895 XSI `#include <ndbm.h>`9896 **DESCRIPTION**9897 The **<ndbm.h>** header shall define the **datum** type as a structure, which shall include at least the  
9898 following members:9899 `void *dptr` A pointer to the application's data.  
9900 `size_t dsize` The size of the object pointed to by *dptr*.9901 The **<ndbm.h>** header shall define the **size\_t** type as described in **<stddef.h>**.9902 The **<ndbm.h>** header shall define the **DBM** type.9903 The **<ndbm.h>** header shall define the following symbolic constants as possible values for the  
9904 *store\_mode* argument to *dbm\_store()*:9905 **DBM\_INSERT** Insertion of new entries only.9906 **DBM\_REPLACE** Allow replacing existing entries.9907 The following shall be declared as functions and may also be defined as macros. Function  
9908 prototypes shall be provided.9909 `int dbm_clearerr(DBM *);`  
9910 `void dbm_close(DBM *);`  
9911 `int dbm_delete(DBM *, datum);`  
9912 `int dbm_error(DBM *);`  
9913 `datum dbm_fetch(DBM *, datum);`  
9914 `datum dbm_firstkey(DBM *);`  
9915 `datum dbm_nextkey(DBM *);`  
9916 `DBM *dbm_open(const char *, int, mode_t);`  
9917 `int dbm_store(DBM *, datum, datum, int);`9918 The **<ndbm.h>** header shall define the **mode\_t** type through **typedef**, as described in  
9919 **<sys/types.h>**.9920 **APPLICATION USAGE**

9921 None.

9922 **RATIONALE**

9923 None.

9924 **FUTURE DIRECTIONS**

9925 None.

9926 **SEE ALSO**9927 **<stddef.h>**, **<sys/types.h>**9928 XSH *dbm\_clearerr()*9929 **CHANGE HISTORY**

9930 First released in Issue 4, Version 2.

*Headers*

&lt;ndbm.h&gt;

- 9931 **Issue 5**  
9932       References to the definitions of **size\_t** and **mode\_t** are added to the DESCRIPTION.
- 9933 **Issue 7**  
9934       This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<net/if.h>**

Headers

9935 **NAME**

9936 net/if.h — sockets local interfaces

9937 **SYNOPSIS**

9938 #include &lt;net/if.h&gt;

9939 **DESCRIPTION**9940 The **<net/if.h>** header shall define the **if\_nameindex** structure, which shall include at least the  
9941 following members:9942 unsigned if\_index Numeric index of the interface.  
9943 char \*if\_name Null-terminated name of the interface.9944 The **<net/if.h>** header shall define the following symbolic constant for the length of a buffer  
9945 containing an interface name (including the terminating NULL character):

9946 IF\_NAMESIZE Interface name length.

9947 The following shall be declared as functions and may also be defined as macros. Function  
9948 prototypes shall be provided.9949 void if\_freenameindex(struct if\_nameindex \*);  
9950 char \*if\_indextoname(unsigned, char \*);  
9951 struct if\_nameindex \*if\_nameindex(void);  
9952 unsigned if\_nametoindex(const char \*);9953 **APPLICATION USAGE**

9954 None.

9955 **RATIONALE**

9956 None.

9957 **FUTURE DIRECTIONS**

9958 None.

9959 **SEE ALSO**9960 XSH [if\\_freenameindex\(\)](#), [if\\_indextoname\(\)](#), [if\\_nameindex\(\)](#), [if\\_nametoindex\(\)](#)9961 **CHANGE HISTORY**

9962 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9963 **Issue 7**

9964 This reference page is clarified with respect to macros and symbolic constants.

9965 **NAME**

9966 netdb.h — definitions for network database operations

9967 **SYNOPSIS**

9968 #include &lt;netdb.h&gt;

9969 **DESCRIPTION**9970 The <netdb.h> header may define the **in\_port\_t** type and the **in\_addr\_t** type as described in  
9971 <netinet/in.h>.9972 The <netdb.h> header shall define the **hostent** structure, which shall include at least the  
9973 following members:

9974	char	*h_name	Official name of the host.
9975	char	**h_aliases	A pointer to an array of pointers to 9976 alternative host names, terminated by a 9977 null pointer.
9978	int	h_addrtype	Address type.
9979	int	h_length	The length, in bytes, of the address.
9980	char	**h_addr_list	A pointer to an array of pointers to network 9981 addresses (in network byte order) for the host, 9982 terminated by a null pointer.

9983 The <netdb.h> header shall define the **netent** structure, which shall include at least the  
9984 following members:

9985	char	*n_name	Official, fully-qualified (including the 9986 domain) name of the host.
9987	char	**n_aliases	A pointer to an array of pointers to 9988 alternative network names, terminated by a 9989 null pointer.
9990	int	n_addrtype	The address type of the network.
9991	uint32_t	n_net	The network number, in host byte order.

9992 The <netdb.h> header shall define the **uint32\_t** type as described in <inttypes.h>.9993 The <netdb.h> header shall define the **protoent** structure, which shall include at least the  
9994 following members:

9995	char	*p_name	Official name of the protocol.
9996	char	**p_aliases	A pointer to an array of pointers to 9997 alternative protocol names, terminated by 9998 a null pointer.
9999	int	p_proto	The protocol number.

10000 The <netdb.h> header shall define the **servent** structure, which shall include at least the  
10001 following members:

10002	char	*s_name	Official name of the service.
10003	char	**s_aliases	A pointer to an array of pointers to 10004 alternative service names, terminated by 10005 a null pointer.
10006	int	s_port	A value which, when converted to <b>uint16_t</b> , 10007 yields the port number in network byte order 10008 at which the service resides.
10009	char	*s_proto	The name of the protocol to use when 10010 contacting the service.

10011 The <netdb.h> header shall define the IPPORT\_RESERVED symbolic constant with the value of  
10012 the highest reserved Internet port number.

### 10013 Address Information Structure

10014 The <netdb.h> header shall define the **addrinfo** structure, which shall include at least the  
10015 following members:

10016	int	ai_flags	Input flags.
10017	int	ai_family	Address family of socket.
10018	int	ai_socktype	Socket type.
10019	int	ai_protocol	Protocol of socket.
10020	socklen_t	ai_addrlen	Length of socket address.
10021	struct sockaddr	*ai_addr	Socket address of socket.
10022	char	*ai_canonname	Canonical name of service location.
10023	struct addrinfo	*ai_next	Pointer to next in list.

10024 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-  
10025 distinct integer constants for use in the *flags* field of the **addrinfo** structure:

10026	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
10027	AI_CANONNAME	Request for canonical name.
10028	AI_NUMERICHOST	Return numeric host address as name.
10029	AI_NUMERICSERV	Inhibit service name resolution.
10030	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them 10031 to the caller as IPv4-mapped IPv6 addresses.
10032	AI_ALL	Query for both IPv4 and IPv6 addresses.
10033	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query 10034 for IPv6 addresses only when an IPv6 address is configured.

10035 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-  
10036 distinct integer constants for use in the *flags* argument to *getnameinfo()*:

10037	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
10038	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
10039	NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
10040	NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
10041	NI_NUMERICSCOPE	
10042		For IPv6 addresses, the numeric form of the scope identifier is returned 10043 instead of its name.
10044	NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

10045 **Address Information Errors**

10046 The <netdb.h> header shall define the following symbolic constants for use as error values for  
 10047 *getaddrinfo()* and *getnameinfo()*. The values shall be suitable for use in *#if* preprocessing  
 10048 directives.

10049 EAI\_AGAIN The name could not be resolved at this time. Future attempts may  
 10050 succeed.

10051 EAI\_BADFLAGS The flags had an invalid value.

10052 EAI\_FAIL A non-recoverable error occurred.

10053 EAI\_FAMILY The address family was not recognized or the address length was invalid  
 10054 for the specified family.

10055 EAI\_MEMORY There was a memory allocation failure.

10056 EAI\_NONAME The name does not resolve for the supplied parameters.

10057 NI\_NAMEREQD is set and the host's name cannot be located, or both  
 10058 *nodename* and *servname* were null.

10059 EAI\_SERVICE The service passed was not recognized for the specified socket type.

10060 EAI\_SOCKTYPE The intended socket type was not recognized.

10061 EAI\_SYSTEM A system error occurred. The error code can be found in *errno*.

10062 EAI\_OVERFLOW An argument buffer overflowed.

10063 The following shall be declared as functions and may also be defined as macros. Function  
 10064 prototypes shall be provided.

10065 void *endhostent*(void);  
 10066 void *endnetent*(void);  
 10067 void *endprotoent*(void);  
 10068 void *endservent*(void);  
 10069 void *freeaddrinfo*(struct *addrinfo* \*);  
 10070 const char \**gai\_strerror*(int);  
 10071 int *getaddrinfo*(const char \*restrict, const char \*restrict,  
 10072 const struct *addrinfo* \*restrict,  
 10073 struct *addrinfo* \*\*restrict);  
 10074 struct *hostent* \**gethostent*(void);  
 10075 int *getnameinfo*(const struct *sockaddr* \*restrict, *socklen\_t*,  
 10076 char \*restrict, *socklen\_t*, char \*restrict,  
 10077 *socklen\_t*, int);  
 10078 struct *netent* \**getnetbyaddr*(uint32\_t, int);  
 10079 struct *netent* \**getnetbyname*(const char \*);  
 10080 struct *netent* \**getnetent*(void);  
 10081 struct *protoent* \**getprotobyname*(const char \*);  
 10082 struct *protoent* \**getprotobynumber*(int);  
 10083 struct *protoent* \**getprotoent*(void);  
 10084 struct *servent* \**getservbyname*(const char \*, const char \*);  
 10085 struct *servent* \**getservbyport*(int, const char \*);  
 10086 struct *servent* \**getservent*(void);  
 10087 void *sethostent*(int);  
 10088 void *setnetent*(int);  
 10089 void *setprotoent*(int);

## &lt;netdb.h&gt;

Headers

10090           void                           setservent(int);

10091           The <netdb.h> header shall define the **socklen\_t** type through **typedef**, as described in  
10092           <sys/socket.h>.

10093           Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,  
10094           <sys/socket.h>, and <inttypes.h>.

10095   **APPLICATION USAGE**

10096           None.

10097   **RATIONALE**

10098           None.

10099   **FUTURE DIRECTIONS**

10100           None.

10101   **SEE ALSO**

10102           <inttypes.h>, <netinet/in.h>, <sys/socket.h>

10103           XSH *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *freeaddrinfo()*, *gai\_strerror()*,  
10104           *getnameinfo()*

10105   **CHANGE HISTORY**

10106           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10107           The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for  
10108           *gai\_strerror()* from **char \*** to **const char \***. This is for coordination with the IPnG Working Group.

10109           IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the  
10110           NI\_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes  
10111           are for alignment with IPv6.

10112   **Issue 7**

10113           SD5-XBD-ERN-14 is applied, changing the description of the *s\_port* member of the **servent**  
10114           structure.

10115           The obsolescent *h\_errno* external integer, and the obsolescent *gethostbyaddr()*, and  
10116           *gethostbyname()* functions are removed, along with the HOST\_NOT\_FOUND, NO\_DATA,  
10117           NO\_RECOVERY, and TRY\_AGAIN macros.

10118           This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;netinet/in.h&gt;

## 10119 NAME

10120           netinet/in.h — Internet address family

## 10121 SYNOPSIS

10122           #include <netinet/in.h>

## 10123 DESCRIPTION

10124           The <netinet/in.h> header shall define the following types:

10125           **in\_port\_t**   Equivalent to the type **uint16\_t** as described in <inttypes.h>.

10126           **in\_addr\_t**   Equivalent to the type **uint32\_t** as described in <inttypes.h>.

10127           The <netinet\_in.h> header shall define the **sa\_family\_t** type as described in <sys/socket.h>.

10128           The <netinet\_in.h> header shall define the **uint8\_t** and **uint32\_t** types as described in <inttypes.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

10131           The <netinet/in.h> header shall define the **in\_addr** structure, which shall include at least the following member:

10133           in\_addr\_t   s\_addr

10134           The <netinet/in.h> header shall define the **sockaddr\_in** structure, which shall include at least the following members:

10136           sa\_family\_t    sin\_family    AF\_INET.  
10137           in\_port\_t       sin\_port     Port number.  
10138           struct in\_addr  sin\_addr     IP address.

10139           The *sin\_port* and *sin\_addr* members shall be in network byte order.

10140           The **sockaddr\_in** structure is used to store addresses for the Internet address family. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

10142 IP6       The <netinet/in.h> header shall define the **in6\_addr** structure, which shall include at least the following member:

10144           uint8\_t   s6\_addr[16]

10145           This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10146           The <netinet/in.h> header shall define the **sockaddr\_in6** structure, which shall include at least the following members:

10148           sa\_family\_t       sin6\_family    AF\_INET6.  
10149           in\_port\_t         sin6\_port     Port number.  
10150           uint32\_t          sin6\_flowinfo  IPv6 traffic class and flow information.  
10151           struct in6\_addr    sin6\_addr     IPv6 address.  
10152           uint32\_t          sin6\_scope\_id  Set of interfaces for a scope.

10153           The *sin6\_port* and *sin6\_addr* members shall be in network byte order.

10154           The **sockaddr\_in6** structure shall be set to zero by an application prior to using it, since implementations are free to have additional, implementation-defined fields in **sockaddr\_in6**.

10156           The *sin6\_scope\_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the scope of the address carried in the *sin6\_addr* field. For a link scope *sin6\_addr*, the application shall ensure that *sin6\_scope\_id* is a link index. For a site scope *sin6\_addr*, the application shall ensure that *sin6\_scope\_id* is a site index. The mapping of *sin6\_scope\_id* to an interface or set of interfaces is implementation-defined.

## &lt;netinet/in.h&gt;

Headers

10161		The <netinet/in.h> header shall declare the following external variable:
10162		<code>const struct in6_addr in6addr_any</code>
10163		This variable is initialized by the system to contain the wildcard IPv6 address. The
10164		<netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
10165		constant at compile time and can be used to initialize a variable of type <b>struct in6_addr</b> to the
10166		IPv6 wildcard address.
10167		The <netinet/in.h> header shall declare the following external variable:
10168		<code>const struct in6_addr in6addr_loopback</code>
10169		This variable is initialized by the system to contain the loopback IPv6 address. The
10170		<netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
10171		constant at compile time and can be used to initialize a variable of type <b>struct in6_addr</b> to the
10172		IPv6 loopback address.
10173		The <netinet/in.h> header shall define the <b>ipv6_mreq</b> structure, which shall include at least the
10174		following members:
10175		<code>struct in6_addr ipv6mr_multiaddr</code> IPv6 multicast address.
10176		<code>unsigned ipv6mr_interface</code> Interface index
10177		The <netinet/in.h> header shall define the following symbolic constants for use as values of the
10178		<i>level</i> argument of <i>getsockopt()</i> and <i>setsockopt()</i> :
10179		IPPROTO_IP Internet protocol.
10180	IP6	IPPROTO_IPV6 Internet Protocol Version 6.
10181		IPPROTO_ICMP Control message protocol.
10182	RS	IPPROTO_RAW Raw IP Packets Protocol.
10183		IPPROTO_TCP Transmission control protocol.
10184		IPPROTO_UDP User datagram protocol.
10185		The <netinet/in.h> header shall define the following symbolic constants for use as destination
10186		addresses for <i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :
10187		INADDR_ANY IPv4 local host address.
10188		INADDR_BROADCAST IPv4 broadcast address.
10189		The <netinet/in.h> header shall define the following symbolic constant, with the value
10190		specified, to help applications declare buffers of the proper size to store IPv4 addresses in string
10191		form:
10192		INET_ADDRSTRLEN 16. Length of the string form for IP.
10193		The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as described in
10194		<arpa/inet.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from
10195		<arpa/inet.h>.
10196	IP6	The <netinet/in.h> header shall define the following symbolic constant, with the value
10197		specified, to help applications declare buffers of the proper size to store IPv6 addresses in string
10198		form:

10199		INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
10200	IP6	The <netinet/in.h> header shall define the following symbolic constants, with distinct integer values, for use in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level IPPROTO_IPV6:	
10201			
10202			
10203		IPV6_JOIN_GROUP	Join a multicast group.
10204		IPV6_LEAVE_GROUP	Quit a multicast group.
10205		IPV6_MULTICAST_HOPS	
10206			Multicast hop limit.
10207		IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10208		IPV6_MULTICAST_LOOP	
10209			Multicast packets are delivered back to the local application.
10210		IPV6_UNICAST_HOPS	Unicast hop limit.
10211		IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10212		The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses. Each macro is of type <b>int</b> and takes a single argument of type <b>const struct in6_addr *</b> :	
10213			
10214		IN6_IS_ADDR_UNSPECIFIED	
10215			Unspecified address.
10216		IN6_IS_ADDR_LOOPBACK	
10217			Loopback address.
10218		IN6_IS_ADDR_MULTICAST	
10219			Multicast address.
10220		IN6_IS_ADDR_LINKLOCAL	
10221			Unicast link-local address.
10222		IN6_IS_ADDR_SITELOCAL	
10223			Unicast site-local address.
10224		IN6_IS_ADDR_V4MAPPED	
10225			IPv4 mapped address.
10226		IN6_IS_ADDR_V4COMPAT	
10227			IPv4-compatible address.
10228		IN6_IS_ADDR_MC_NODELOCAL	
10229			Multicast node-local address.
10230		IN6_IS_ADDR_MC_LINKLOCAL	
10231			Multicast link-local address.
10232		IN6_IS_ADDR_MC_SITELOCAL	
10233			Multicast site-local address.
10234		IN6_IS_ADDR_MC_ORGLOCAL	
10235			Multicast organization-local address.
10236		IN6_IS_ADDR_MC_GLOBAL	
10237			Multicast global address.

## &lt;netinet/in.h&gt;

Headers

## 10238 APPLICATION USAGE

10239 None.

## 10240 RATIONALE

10241 None.

## 10242 FUTURE DIRECTIONS

10243 None.

## 10244 SEE ALSO

10245 [Section 4.9](#) (on page 110), [<arpa/inet.h>](#), [<inttypes.h>](#), [<sys/socket.h>](#)10246 XSH [connect\(\)](#), [getsockopt\(\)](#), [htonl\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#)

## 10247 CHANGE HISTORY

10248 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10249 The *sin\_zero* member was removed from the `sockaddr_in` structure as per The Open Group Base  
10250 Resolution bwg2001-004.10251 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding `const` qualifiers to  
10252 the *in6addr\_any* and *in6addr\_loopback* external variables.10253 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which  
10254 structure members are in network byte order.

## 10255 Issue 7

10256 This reference page is clarified with respect to macros and symbolic constants.

10257 **NAME**

10258           netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10259 **SYNOPSIS**

10260           #include <netinet/tcp.h>

10261 **DESCRIPTION**

10262           The <netinet/tcp.h> header shall define the following symbolic constant for use as a socket  
10263           option at the IPPROTO\_TCP level:

10264           TCP\_NODELAY   Avoid coalescing of small segments.

10265           The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved  
10266           via *getsockopt()*.

10267 **APPLICATION USAGE**

10268           None.

10269 **RATIONALE**

10270           None.

10271 **FUTURE DIRECTIONS**

10272           None.

10273 **SEE ALSO**

10274           <sys/socket.h>

10275           XSH *getsockopt()*, *setsockopt()*

10276 **CHANGE HISTORY**

10277           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10278 **Issue 7**

10279           This reference page is clarified with respect to macros and symbolic constants.

**<nl\_types.h>**

Headers

10280 **NAME**

10281 nl\_types.h — data types

10282 **SYNOPSIS**

10283 #include &lt;nl\_types.h&gt;

10284 **DESCRIPTION**

10285 The &lt;nl\_types.h&gt; header shall define at least the following types:

10286 **nl\_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*  
 10287 to identify a catalog descriptor.

10288 **nl\_item** Used by *nl\_langinfo()* to identify items of *langinfo* data. Values of objects  
 10289 of type **nl\_item** are defined in <langinfo.h>.

10290 The &lt;nl\_types.h&gt; header shall define at least the following symbolic constants:

10291 **NL\_SETD** Used by *gencat* when no *\$set* directive is specified in a message text source  
 10292 file. This constant can be passed as the value of *set\_id* on subsequent calls  
 10293 to *catgets()* (that is, to retrieve messages from the default message set).  
 10294 The value of **NL\_SETD** is implementation-defined.

10295 **NL\_CAT\_LOCALE** Value that must be passed as the *oflag* argument to *catopen()* to ensure that  
 10296 message catalog selection depends on the *LC\_MESSAGES* locale category,  
 10297 rather than directly on the *LANG* environment variable.

10298 The following shall be declared as functions and may also be defined as macros. Function  
10299 prototypes shall be provided.

```
10300 int      catclose(nl_catd);
10301 char    *catgets(nl_catd, int, int, const char *);
10302 nl_catd  catopen(const char *, int);
```

10303 **APPLICATION USAGE**

10304 None.

10305 **RATIONALE**

10306 None.

10307 **FUTURE DIRECTIONS**

10308 None.

10309 **SEE ALSO**

10310 &lt;langinfo.h&gt;

10311 XSH *catclose()*, *catgets()*, *catopen()*, *nl\_langinfo()*10312 XCU *gencat*10313 **CHANGE HISTORY**

10314 First released in Issue 2.

10315 **Issue 7**

10316 The &lt;nl\_types.h&gt; header is moved from the XSI option to the Base.

10317 This reference page is clarified with respect to macros and symbolic constants.

10318 **NAME**

10319 poll.h — definitions for the poll() function

10320 **SYNOPSIS**

10321 #include &lt;poll.h&gt;

10322 **DESCRIPTION**10323 The <poll.h> header shall define the **pollfd** structure, which shall include at least the following  
10324 members:

10325 int fd The following descriptor being polled.

10326 short events The input event flags (see below).

10327 short revents The output event flags (see below).

10328 The <poll.h> header shall define the following type through **typedef**:10329 **nfds\_t** An unsigned integer type used for the number of file descriptors.10330 The implementation shall support one or more programming environments in which the width  
10331 of **nfds\_t** is no greater than the width of type **long**. The names of these programming  
10332 environments can be obtained using the *confstr()* function or the *getconf* utility.10333 The <poll.h> header shall define the following symbolic constants, zero or more of which may  
10334 be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

10335 POLLIN Data other than high-priority data may be read without blocking.

10336 POLLRDNORM Normal data may be read without blocking.

10337 POLLRDBAND Priority data may be read without blocking.

10338 POLLPRI High priority data may be read without blocking.

10339 POLLOUT Normal data may be written without blocking.

10340 POLLWRNORM Equivalent to POLLOUT.

10341 POLLWRBAND Priority data may be written.

10342 POLLERR An error has occurred (*revents* only).10343 POLLHUP Device has been disconnected (*revents* only).10344 POLLNVAL Invalid *fd* member (*revents* only).10345 The significance and semantics of normal, priority, and high-priority data are file and device-  
10346 specific.10347 The following shall be declared as a function and may also be defined as a macro. A function  
10348 prototype shall be provided.

10349 int poll(struct pollfd [], nfds\_t, int);

10350 **APPLICATION USAGE**

10351 None.

10352 **RATIONALE**

10353 None.

10354 **FUTURE DIRECTIONS**

10355 None.

## &lt;poll.h&gt;

Headers

10356 **SEE ALSO**10357 XSH *confstr()*, *poll()*10358 XCU *getconf*10359 **CHANGE HISTORY**

10360 First released in Issue 4, Version 2.

10361 **Issue 6**10362 The description of the symbolic constants is updated to match the *poll()* function.10363 Text related to STREAMS has been moved to the *poll()* reference page.10364 A note is added to the DESCRIPTION regarding the significance and semantics of normal,  
10365 priority, and high-priority data.10366 **Issue 7**

10367 The &lt;poll.h&gt; header is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;pthread.h&gt;

10368 **NAME**

10369 pthread.h — threads

10370 **SYNOPSIS**

10371 #include &lt;pthread.h&gt;

10372 **DESCRIPTION**

10373 The &lt;pthread.h&gt; header shall define the following symbolic constants:

10374 PTHREAD\_BARRIER\_SERIAL\_THREAD  
 10375 PTHREAD\_CANCEL\_ASYNCHRONOUS  
 10376 PTHREAD\_CANCEL\_ENABLE  
 10377 PTHREAD\_CANCEL\_DEFERRED  
 10378 PTHREAD\_CANCEL\_DISABLE  
 10379 PTHREAD\_CANCELED  
 10380 PTHREAD\_CREATE\_DETACHED  
 10381 PTHREAD\_CREATE\_JOINABLE  
 10382 TPS PTHREAD\_EXPLICIT\_SCHED  
 10383 PTHREAD\_INHERIT\_SCHED  
 10384 PTHREAD\_MUTEX\_DEFAULT  
 10385 PTHREAD\_MUTEX\_ERRORCHECK  
 10386 PTHREAD\_MUTEX\_NORMAL  
 10387 PTHREAD\_MUTEX\_RECURSIVE  
 10388 PTHREAD\_MUTEX\_ROBUST  
 10389 PTHREAD\_MUTEX\_STALLED  
 10390 PTHREAD\_ONCE\_INIT  
 10391 RPI|TPI PTHREAD\_PRIO\_INHERIT  
 10392 MC1 PTHREAD\_PRIO\_NONE  
 10393 RPP|TPP PTHREAD\_PRIO\_PROTECT  
 10394 PTHREAD\_PROCESS\_SHARED  
 10395 PTHREAD\_PROCESS\_PRIVATE  
 10396 TPS PTHREAD\_SCOPE\_PROCESS  
 10397 PTHREAD\_SCOPE\_SYSTEM

10398 The <pthread.h> header shall define the following compile-time constant expressions valid as  
 10399 initializers for the following types:

Name	Initializer for Type
PTHREAD_COND_INITIALIZER	pthread_cond_t
PTHREAD_MUTEX_INITIALIZER	pthread_mutex_t
PTHREAD_RWLOCK_INITIALIZER	pthread_rwlock_t

10404 The <pthread.h> header shall define the pthread\_attr\_t, pthread\_barrier\_t,  
 10405 pthread\_barrierattr\_t, pthread\_cond\_t, pthread\_condattr\_t, pthread\_key\_t, pthread\_mutex\_t,  
 10406 pthread\_mutexattr\_t, pthread\_once\_t, pthread\_rwlock\_t, pthread\_rwlockattr\_t,  
 10407 pthread\_spinlock\_t, and pthread\_t types as described in <sys/types.h>.

10408 The following shall be declared as functions and may also be defined as macros. Function  
 10409 prototypes shall be provided.

```
10410 int pthread_atfork(void (*)(void), void (*)(void),
10411                  void (*)(void));
10412 int pthread_attr_destroy(pthread_attr_t *);
10413 int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10414 int pthread_attr_getguardsize(const pthread_attr_t *restrict,
```

## &lt;pthread.h&gt;

## Headers

```

10415         size_t *restrict);
10416 TPS    int  pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10417         int *restrict);
10418         int  pthread_attr_getschedparam(const pthread_attr_t *restrict,
10419         struct sched_param *restrict);
10420 TPS    int  pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10421         int *restrict);
10422         int  pthread_attr_getscope(const pthread_attr_t *restrict,
10423         int *restrict);
10424 TSA TSS int  pthread_attr_getstack(const pthread_attr_t *restrict,
10425         void **restrict, size_t *restrict);
10426 TSS    int  pthread_attr_getstacksize(const pthread_attr_t *restrict,
10427         size_t *restrict);
10428         int  pthread_attr_init(pthread_attr_t *);
10429         int  pthread_attr_setdetachstate(pthread_attr_t *, int);
10430         int  pthread_attr_setguardsize(pthread_attr_t *, size_t);
10431 TPS    int  pthread_attr_setinheritsched(pthread_attr_t *, int);
10432         int  pthread_attr_setschedparam(pthread_attr_t *restrict,
10433         const struct sched_param *restrict);
10434 TPS    int  pthread_attr_setschedpolicy(pthread_attr_t *, int);
10435         int  pthread_attr_setscope(pthread_attr_t *, int);
10436 TSA TSS int  pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10437 TSS    int  pthread_attr_setstacksize(pthread_attr_t *, size_t);
10438         int  pthread_barrier_destroy(pthread_barrier_t *);
10439         int  pthread_barrier_init(pthread_barrier_t *restrict,
10440         const pthread_barrierattr_t *restrict, unsigned);
10441         int  pthread_barrier_wait(pthread_barrier_t *);
10442         int  pthread_barrierattr_destroy(pthread_barrierattr_t *);
10443 TSH    int  pthread_barrierattr_getpshared(
10444         const pthread_barrierattr_t *restrict, int *restrict);
10445         int  pthread_barrierattr_init(pthread_barrierattr_t *);
10446 TSH    int  pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10447         int  pthread_cancel(pthread_t);
10448         void pthread_cleanup_pop(int);
10449         void pthread_cleanup_push(void (*)(void*), void *);
10450         int  pthread_cond_broadcast(pthread_cond_t *);
10451         int  pthread_cond_destroy(pthread_cond_t *);
10452         int  pthread_cond_init(pthread_cond_t *restrict,
10453         const pthread_condattr_t *restrict);
10454         int  pthread_cond_signal(pthread_cond_t *);
10455         int  pthread_cond_timedwait(pthread_cond_t *restrict,
10456         pthread_mutex_t *restrict, const struct timespec *restrict);
10457         int  pthread_cond_wait(pthread_cond_t *restrict,
10458         pthread_mutex_t *restrict);
10459         int  pthread_condattr_destroy(pthread_condattr_t *);
10460         int  pthread_condattr_getclock(const pthread_condattr_t *restrict,
10461         clockid_t *restrict);
10462 TSH    int  pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10463         int *restrict);
10464         int  pthread_condattr_init(pthread_condattr_t *);
10465         int  pthread_condattr_setclock(pthread_condattr_t *, clockid_t);

```

## Headers

## &lt;pthread.h&gt;

```

10466 TSH      int  pthread_condattr_setpshared(pthread_condattr_t *, int);
10467         int  pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10468             void *(*)(void*), void *restrict);
10469         int  pthread_detach(pthread_t);
10470         int  pthread_equal(pthread_t, pthread_t);
10471         void pthread_exit(void *);
10472 OB XSI     int  pthread_getconcurrency(void);
10473 TCT        int  pthread_getcpuclockid(pthread_t, clockid_t *);
10474 TPS        int  pthread_getschedparam(pthread_t, int *restrict,
10475             struct sched_param *restrict);
10476         void *pthread_getspecific(pthread_key_t);
10477         int  pthread_join(pthread_t, void **);
10478         int  pthread_key_create(pthread_key_t *, void (*)(void*));
10479         int  pthread_key_delete(pthread_key_t);
10480         int  pthread_mutex_consistent(pthread_mutex_t *);
10481         int  pthread_mutex_destroy(pthread_mutex_t *);
10482 RPP|TPP   int  pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10483             int *restrict);
10484         int  pthread_mutex_init(pthread_mutex_t *restrict,
10485             const pthread_mutexattr_t *restrict);
10486         int  pthread_mutex_lock(pthread_mutex_t *);
10487 RPP|TPP   int  pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10488             int *restrict);
10489         int  pthread_mutex_timedlock(pthread_mutex_t *restrict,
10490             const struct timespec *restrict);
10491         int  pthread_mutex_trylock(pthread_mutex_t *);
10492         int  pthread_mutex_unlock(pthread_mutex_t *);
10493         int  pthread_mutexattr_destroy(pthread_mutexattr_t *);
10494 RPP|TPP   int  pthread_mutexattr_getprioceiling(
10495             const pthread_mutexattr_t *restrict, int *restrict);
10496 MC1        int  pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10497             int *restrict);
10498 TSH        int  pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10499             int *restrict);
10500         int  pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
10501             int *restrict);
10502         int  pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10503             int *restrict);
10504         int  pthread_mutexattr_init(pthread_mutexattr_t *);
10505 RPP|TPP   int  pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10506 MC1        int  pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10507 TSH        int  pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10508         int  pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
10509         int  pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10510         int  pthread_once(pthread_once_t *, void (*)(void));
10511         int  pthread_rwlock_destroy(pthread_rwlock_t *);
10512         int  pthread_rwlock_init(pthread_rwlock_t *restrict,
10513             const pthread_rwlockattr_t *restrict);
10514         int  pthread_rwlock_rdlock(pthread_rwlock_t *);
10515         int  pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10516             const struct timespec *restrict);
10517         int  pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,

```

**<pthread.h>**

Headers

```

10518         const struct timespec *restrict);
10519 int pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10520 int pthread_rwlock_trywrlock(pthread_rwlock_t *);
10521 int pthread_rwlock_unlock(pthread_rwlock_t *);
10522 int pthread_rwlock_wrlock(pthread_rwlock_t *);
10523 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10524 TSH int pthread_rwlockattr_getpshared(
10525     const pthread_rwlockattr_t *restrict, int *restrict);
10526 int pthread_rwlockattr_init(pthread_rwlockattr_t *);
10527 TSH int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10528 pthread_t
10529     pthread_self(void);
10530 int pthread_setcancelstate(int, int *);
10531 int pthread_setcanceltype(int, int *);
10532 OB XSI int pthread_setconcurrency(int);
10533 TPS int pthread_setschedparam(pthread_t, int,
10534     const struct sched_param *);
10535 int pthread_setschedprio(pthread_t, int);
10536 int pthread_setspecific(pthread_key_t, const void *);
10537 int pthread_spin_destroy(pthread_spinlock_t *);
10538 int pthread_spin_init(pthread_spinlock_t *, int);
10539 int pthread_spin_lock(pthread_spinlock_t *);
10540 int pthread_spin_trylock(pthread_spinlock_t *);
10541 int pthread_spin_unlock(pthread_spinlock_t *);
10542 void pthread_testcancel(void);

```

10543 Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and  
10544 **<time.h>** visible.

**10545 APPLICATION USAGE**

10546 None.

**10547 RATIONALE**

10548 None.

**10549 FUTURE DIRECTIONS**

10550 None.

**10551 SEE ALSO**

10552 **<sched.h>**, **<sys/types.h>**, **<time.h>**

10553 XSH *pthread\_atfork()*, *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*,  
10554 *pthread\_attr\_getguardsize()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*,  
10555 *pthread\_attr\_getschedpolicy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getstack()*,  
10556 *pthread\_attr\_getstacksize()*, *pthread\_barrier\_destroy()*, *pthread\_barrier\_wait()*,  
10557 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_cancel()*,  
10558 *pthread\_cleanup\_pop()*, *pthread\_cond\_broadcast()*, *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*,  
10559 *pthread\_condattr\_destroy()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*,  
10560 *pthread\_create()*, *pthread\_detach()*, *pthread\_equal()*, *pthread\_exit()*, *pthread\_getconcurrency()*,  
10561 *pthread\_getcpuclockid()*, *pthread\_getschedparam()*, *pthread\_getspecific()*, *pthread\_join()*,  
10562 *pthread\_key\_create()*, *pthread\_key\_delete()*, *pthread\_mutex\_consistent()*, *pthread\_mutex\_destroy()*,  
10563 *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*,  
10564 *pthread\_mutexattr\_destroy()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
10565 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_getrobust()*, *pthread\_mutexattr\_gettype()*,  
10566 *pthread\_once()*, *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,

10567 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
 10568 *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_self()*,  
 10569 *pthread\_setcancelstate()*, *pthread\_setschedprio()*, *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*,  
 10570 *pthread\_spin\_unlock()*

#### 10571 CHANGE HISTORY

10572 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 10573 Issue 6

10574 The RTT margin markers are broken out into their POSIX options.

10575 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
 10576 *pthread\_cond\_wait()* function.

10577 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
 10578 *pthread\_setschedparam()* function so that its second argument is of type **int**.

10579 The *pthread\_getcpuclockid()* and *pthread\_mutex\_timedlock()* functions are added for alignment  
 10580 with IEEE Std 1003.1d-1999.

10581 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10582 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*,  
 10583 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*,  
 10584 *pthread\_barrierattr\_setpshared()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_setclock()*,  
 10585 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_spin\_destroy()*,  
 10586 *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, and *pthread\_spin\_unlock()*.

10587 PTHREAD\_RWLOCK\_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

10588 Functions previously marked as part of the Read-Write Locks option are now moved to the  
 10589 Threads option.

10590 The **restrict** keyword is added to the prototypes for *pthread\_attr\_getguardsize()*,  
 10591 *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*, *pthread\_attr\_getschedpolicy()*,  
 10592 *pthread\_attr\_getscope()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*,  
 10593 *pthread\_attr\_setschedparam()*, *pthread\_barrier\_init()*, *pthread\_barrierattr\_getpshared()*,  
 10594 *pthread\_cond\_init()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
 10595 *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
 10596 *pthread\_getschedparam()*, *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_init()*,  
 10597 *pthread\_mutex\_setprioceiling()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
 10598 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_gettype()*, *pthread\_rwlock\_init()*,  
 10599 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlockattr\_getpshared()*, and  
 10600 *pthread\_sigmask()*.

10601 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and  
 10602 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI option.

10603 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for  
 10604 the *pthread\_kill()* and *pthread\_sigmask()* functions. These are required to be in the <signal.h>  
 10605 header. They are allowed here through the name space rules.

10606 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread\_setschedprio()* function.

10607 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors  
 10608 that were in contradiction with the System Interfaces volume of POSIX.1-2008.

## &lt;pthread.h&gt;

Headers

10609 **Issue 7**

10610 SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread\_mutex\_timedlock()*  
10611 function prototype.

10612 SD5-XBD-ERN-62 is applied.

10613 Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the  
10614 PTHREAD\_RWLOCK\_INITIALIZER symbol.

10615 The <pthread.h> header is moved from the Threads option to the Base.

10616 The following extended mutex types are moved from the XSI option to the Base:

10617 PTHREAD\_MUTEX\_NORMAL  
10618 PTHREAD\_MUTEX\_ERRORCHECK  
10619 PTHREAD\_MUTEX\_RECURSIVE  
10620 PTHREAD\_MUTEX\_DEFAULT

10621 The PTHREAD\_MUTEX\_ROBUST and PTHREAD\_MUTEX\_STALLED symbols and the  
10622 *pthread\_mutex\_consistent()*, *pthread\_mutexattr\_getrobust()*, and *pthread\_mutexattr\_setrobust()*  
10623 functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

10624 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options  
10625 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex  
10626 or Robust Mutex Priority Inheritance, respectively.

10627 This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;pwd.h&gt;

10628 **NAME**

10629           pwd.h — password structure

10630 **SYNOPSIS**

10631           #include &lt;pwd.h&gt;

10632 **DESCRIPTION**10633           The <pwd.h> header shall define the **struct passwd**, structure, which shall include at least the  
10634           following members:

10635	char	*pw_name	User's login name.
10636	uid_t	pw_uid	Numerical user ID.
10637	gid_t	pw_gid	Numerical group ID.
10638	char	*pw_dir	Initial working directory.
10639	char	*pw_shell	Program to use as shell.

10640           The <pwd.h> header shall define the **gid\_t**, **uid\_t**, and **size\_t** types as described in  
10641           <sys/types.h>.10642           The following shall be declared as functions and may also be defined as macros. Function  
10643           prototypes shall be provided.

```

10644 XSI void      endpwent(void);
10645 struct passwd *getpwent(void);
10646 struct passwd *getpwnam(const char *);
10647 int      getpwnam_r(const char *, struct passwd *, char *,
10648                  size_t, struct passwd **);
10649 struct passwd *getpwuid(uid_t);
10650 int      getpwuid_r(uid_t, struct passwd *, char *,
10651                  size_t, struct passwd **);
10652 XSI void      setpwent(void);

```

10653 **APPLICATION USAGE**

10654           None.

10655 **RATIONALE**

10656           None.

10657 **FUTURE DIRECTIONS**

10658           None.

10659 **SEE ALSO**

10660           &lt;sys/types.h&gt;

10661           XSH *endpwent()*, *getpwnam()*, *getpwuid()*10662 **CHANGE HISTORY**

10663           First released in Issue 1.

10664 **Issue 5**

10665           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10666 **Issue 6**10667           The following new requirements on POSIX implementations derive from alignment with the  
10668           Single UNIX Specification:

- 10669
- The **gid\_t** and **uid\_t** types are mandated.

**<pwd.h>***Headers*

10670                   • The *getpwnam\_r()* and *getpwuid\_r()* functions are marked as part of the Thread-Safe  
10671                    Functions option.

10672 **Issue 7**

10673                   SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size\_t** type.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

10674 **NAME**

10675 regex.h — regular expression matching types

10676 **SYNOPSIS**

10677 #include &lt;regex.h&gt;

10678 **DESCRIPTION**10679 The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,  
10680 *regex()*, *regerror()*, and *regfree()* functions.10681 The <regex.h> header shall define the **regex\_t** structure type, which shall include at least the  
10682 following member:

10683 size\_t re\_nsub Number of parenthesized subexpressions.

10684 The <regex.h> header shall define the **size\_t** type as described in <sys/types.h>.10685 The <regex.h> header shall define the **regoff\_t** type as a signed integer type that can hold the  
10686 largest value that can be stored in either a **ptrdiff\_t** type or a **ssize\_t** type.10687 The <regex.h> header shall define the **regmatch\_t** structure type, which shall include at least the  
10688 following members:10689 regoff\_t rm\_so Byte offset from start of string  
10690 to start of substring.10691 regoff\_t rm\_eo Byte offset from start of string of the  
10692 first character after the end of substring.10693 The <regex.h> header shall define the following symbolic constants for the *cflags* parameter to  
10694 the *regcomp()* function:

10695 REG\_EXTENDED Use Extended Regular Expressions.

10696 REG\_ICASE Ignore case in match.

10697 REG\_NOSUB Report only success or fail in *regex()*.

10698 REG\_NEWLINE Change the handling of &lt;newline&gt;.

10699 The <regex.h> header shall define the following symbolic constants for the *eflags* parameter to  
10700 the *regex()* function:10701 REG\_NOTBOL The <circumflex> character ('^'), when taken as a special character, does  
10702 not match the beginning of *string*.10703 REG\_NOTEOL The <dollar-sign> ('\$'), when taken as a special character, does not  
10704 match the end of *string*.

10705 The &lt;regex.h&gt; header shall define the following symbolic constants as error return values:

10706 REG\_NOMATCH *regex()* failed to match.

10707 REG\_BADPAT Invalid regular expression.

10708 REG\_ECOLLATE Invalid collating element referenced.

10709 REG\_ETYPE Invalid character class type referenced.

10710 REG\_ESCAPE Trailing &lt;backslash&gt; character in pattern.

10711 REG\_ESUBREG Number in *\digit* invalid or in error.

**<regex.h>**

Headers

10712	REG_EBRACK	"[]" imbalance.
10713	REG_EPAREN	"\(\)" or "()" imbalance.
10714	REG_EBRACE	"\{\}" imbalance.
10715	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than two numbers, first larger than second.
10716		
10717	REG_ERANGE	Invalid endpoint in range expression.
10718	REG_ESPACE	Out of memory.
10719	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
10720		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
10721		
10722		<code>int regcomp(regex_t *restrict, const char *restrict, int);</code>
10723		<code>size_t regerror(int, const regex_t *restrict, char *restrict, size_t);</code>
10724		<code>int regexec(const regex_t *restrict, const char *restrict, size_t, regmatch_t [restrict], int);</code>
10725		<code>regmatch_t [restrict], int);</code>
10726		<code>void regfree(regex_t *);</code>
10727		The implementation may define additional macros or constants using names beginning with REG_.
10728		
10729	<b>APPLICATION USAGE</b>	
10730		None.
10731	<b>RATIONALE</b>	
10732		None.
10733	<b>FUTURE DIRECTIONS</b>	
10734		None.
10735	<b>SEE ALSO</b>	
10736		<a href="#">&lt;sys/types.h&gt;</a>
10737		XSH <code>regcomp()</code>
10738	<b>CHANGE HISTORY</b>	
10739		First released in Issue 4.
10740		Originally derived from the ISO POSIX-2 standard.
10741	<b>Issue 6</b>	
10742		The REG_ENOSYS constant is marked obsolescent.
10743		The <b>restrict</b> keyword is added to the prototypes for <code>regcomp()</code> , <code>regerror()</code> , and <code>regexec()</code> .
10744		A statement is added that the <b>size_t</b> type is defined as described in <a href="#">&lt;sys/types.h&gt;</a> .
10745	<b>Issue 7</b>	
10746		SD5-XBD-ERN-60 is applied.
10747		The obsolescent REG_ENOSYS constant is removed.
10748		This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;sched.h&gt;

10749 **NAME**

10750 sched.h — execution scheduling

10751 **SYNOPSIS**

10752 #include <sched.h>

10753 **DESCRIPTION**

10754 PS The <sched.h> header shall define the **pid\_t** type as described in <sys/types.h>.

10755 SS|TSP The <sched.h> header shall define the **time\_t** type as described in <sys/types.h>.

10756 The <sched.h> header shall define the **timespec** structure as described in <time.h>.

10757 The <sched.h> header shall define the **sched\_param** structure, which shall include the scheduling parameters required for implementation of each supported scheduling policy. This structure shall include at least the following member:

10760 int sched\_priority Process or thread execution scheduling priority.

10761 SS|TSP The **sched\_param** structure defined in <sched.h> shall include the following members in addition to those specified above:

10763 int sched\_ss\_low\_priority Low scheduling priority for sporadic server.

10764 struct timespec sched\_ss\_repl\_period Replenishment period for sporadic server.

10765 struct timespec sched\_ss\_init\_budget Initial budget for sporadic server.

10766 int sched\_ss\_max\_repl Maximum pending replenishments for sporadic server.

10770 Each process or thread is controlled by an associated scheduling policy and priority. Associated with each policy is a priority range. Each policy definition specifies the minimum priority range for that policy. The priority ranges for each policy may overlap the priority ranges of other policies.

10774 Four scheduling policies are defined; others may be defined by the implementation. The four standard policies are indicated by the values of the following symbolic constants:

10776 PS|TPS **SCHED\_FIFO** First in-first out (FIFO) scheduling policy.

10777 PS|TPS **SCHED\_RR** Round robin scheduling policy.

10778 SS|TSP **SCHED\_SPORADIC** Sporadic server scheduling policy.

10779 PS|TPS **SCHED\_OTHER** Another scheduling policy.

10780 The values of these constants are distinct.

10781 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

10783 PS|TPS int sched\_get\_priority\_max(int);

10784 int sched\_get\_priority\_min(int);

10785 PS int sched\_getparam(pid\_t, struct sched\_param \*);

10786 int sched\_getscheduler(pid\_t);

10787 PS|TPS int sched\_rr\_get\_interval(pid\_t, struct timespec \*);

10788 PS int sched\_setparam(pid\_t, const struct sched\_param \*);

10789 int sched\_setscheduler(pid\_t, int, const struct sched\_param \*);

10790 int sched\_yield(void);

## &lt;sched.h&gt;

Headers

- 10791 Inclusion of the <sched.h> header may make visible all symbols from the <time.h> header.
- 10792 **APPLICATION USAGE**
- 10793 None.
- 10794 **RATIONALE**
- 10795 None.
- 10796 **FUTURE DIRECTIONS**
- 10797 None.
- 10798 **SEE ALSO**
- 10799 <sys/types.h>, <time.h>
- 10800 XSH *sched\_get\_priority\_max()*, *sched\_getparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
10801 *sched\_setparam()*, *sched\_setscheduler()*, *sched\_yield()*
- 10802 **CHANGE HISTORY**
- 10803 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 10804 **Issue 6**
- 10805 The <sched.h> header is marked as part of the Process Scheduling option.
- 10806 Sporadic server members are added to the **sched\_param** structure, and the SCHED\_SPORADIC  
10807 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
- 10808 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched\_param** structure whose  
10809 members *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* should be type **struct timespec** and not  
10810 **timespec**.
- 10811 Symbols from <time.h> may be made visible when <sched.h> is included.
- 10812 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,  
10813 aligning the function prototype shading and margin codes with the System Interfaces volume of  
10814 IEEE Std 1003.1-2001.
- 10815 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the  
10816 DESCRIPTION to differentiate between thread and process execution.
- 10817 **Issue 7**
- 10818 SD5-XBD-ERN-13 is applied.
- 10819 Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the options markings.
- 10820 The <sched.h> headers is moved from the Threads option to the Base.
- 10821 Declarations for the **pid\_t** and **time\_t** types and the **timespec** structure are added.

## Headers

## &lt;search.h&gt;

10822 **NAME**

10823 search.h — search tables

10824 **SYNOPSIS**10825 XSI `#include <search.h>`10826 **DESCRIPTION**10827 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the  
10828 following members:10829 char \*key  
10830 void \*data10831 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as  
10832 follows:10833 enum { FIND, ENTER } ACTION;  
10834 enum { preorder, postorder, endorder, leaf } VISIT;10835 The <search.h> header shall define the **size\_t** type as described in <sys/types.h>.10836 The following shall be declared as functions and may also be defined as macros. Function  
10837 prototypes shall be provided.10838 int hcreate(size\_t);  
10839 void hdestroy(void);  
10840 ENTRY \*hsearch(ENTRY, ACTION);  
10841 void insque(void \*, void \*);  
10842 void \*lfind(const void \*, const void \*, size\_t \*,  
10843 size\_t, int (\*)(const void \*, const void \*));  
10844 void \*lsearch(const void \*, void \*, size\_t \*,  
10845 size\_t, int (\*)(const void \*, const void \*));  
10846 void remque(void \*);  
10847 void \*tdelete(const void \*restrict, void \*\*restrict,  
10848 int (\*)(const void \*, const void \*));  
10849 void \*tfind(const void \*, void \*const \*,  
10850 int (\*)(const void \*, const void \*));  
10851 void \*tsearch(const void \*, void \*\*,  
10852 int (\*)(const void \*, const void \*));  
10853 void twalk(const void \*,  
10854 void (\*)(const void \*, VISIT, int ));10855 **APPLICATION USAGE**

10856 None.

10857 **RATIONALE**

10858 None.

10859 **FUTURE DIRECTIONS**

10860 None.

10861 **SEE ALSO**10862 [<sys/types.h>](#)10863 XSH [hcreate\(\)](#), [insque\(\)](#), [lsearch\(\)](#), [tdelete\(\)](#)

10864 **CHANGE HISTORY**

10865 First released in Issue 1. Derived from Issue 1 of the SVID.

10866 **Issue 6**

10867 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and *tsearch()*.

10869 The **restrict** keyword is added to the prototype for *tdelete()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

10870 **NAME**

10871 semaphore.h — semaphores

10872 **SYNOPSIS**

10873 #include &lt;semaphore.h&gt;

10874 **DESCRIPTION**

10875 The <semaphore.h> header shall define the **sem\_t** type, used in performing semaphore  
 10876 operations. The semaphore may be implemented using a file descriptor, in which case  
 10877 applications are able to open up at least a total of {OPEN\_MAX} files and semaphores.

10878 The <semaphore.h> header shall define the symbolic constant SEM\_FAILED which shall have  
 10879 type **sem\_t\***.

10880 The following shall be declared as functions and may also be defined as macros. Function  
 10881 prototypes shall be provided.

```
10882 int sem_close(sem_t *);
10883 int sem_destroy(sem_t *);
10884 int sem_getvalue(sem_t *restrict, int *restrict);
10885 int sem_init(sem_t *, int, unsigned);
10886 sem_t *sem_open(const char *, int, ...);
10887 int sem_post(sem_t *);
10888 int sem_timedwait(sem_t *restrict, const struct timespec *restrict);
10889 int sem_trywait(sem_t *);
10890 int sem_unlink(const char *);
10891 int sem_wait(sem_t *);
```

10892 Inclusion of the <semaphore.h> header may make visible symbols defined in the <fcntl.h> and  
 10893 <time.h> headers.

10894 **APPLICATION USAGE**

10895 None.

10896 **RATIONALE**

10897 None.

10898 **FUTURE DIRECTIONS**

10899 None.

10900 **SEE ALSO**

10901 &lt;fcntl.h&gt;, &lt;sys/types.h&gt;, &lt;time.h&gt;

10902 XSH *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
 10903 *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*

10904 **CHANGE HISTORY**

10905 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10906 **Issue 6**

10907 The &lt;semaphore.h&gt; header is marked as part of the Semaphores option.

10908 The Open Group Corrigendum U021/3 is applied, adding a description of SEM\_FAILED.

10909 The *sem\_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10910 The **restrict** keyword is added to the prototypes for *sem\_getvalue()* and *sem\_timedwait()*.

**<semaphore.h>***Headers*10911 **Issue 7**

10912 SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>**  
10913 header.

10914 The **<semaphore.h>** header is moved from the Semaphores option to the Base.

10915 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;setjmp.h&gt;

10916 **NAME**

10917 setjmp.h — stack environment declarations

10918 **SYNOPSIS**

10919 #include <setjmp.h>

10920 **DESCRIPTION**

10921 CX Some of the functionality described on this reference page extends the ISO C standard.  
 10922 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 10923 enable the visibility of these symbols in this header.

10924 CX The <setjmp.h> header shall define the array types **jmp\_buf** and **sigjmp\_buf**.

10925 The following shall be declared as functions and may also be defined as macros. Function  
 10926 prototypes shall be provided.

10927 OB XSI void \_longjmp(jmp\_buf, int);

10928 void longjmp(jmp\_buf, int);

10929 CX void siglongjmp(sigjmp\_buf, int);

10930 The following may be declared as functions, or defined as macros, or both. If functions are  
 10931 declared, function prototypes shall be provided.

10932 OB XSI int \_setjmp(jmp\_buf);

10933 int setjmp(jmp\_buf);

10934 CX int sigsetjmp(sigjmp\_buf, int);

10935 **APPLICATION USAGE**

10936 None.

10937 **RATIONALE**

10938 None.

10939 **FUTURE DIRECTIONS**

10940 None.

10941 **SEE ALSO**

10942 XSH Section 2.2 (on page 468), *\_longjmp()*, *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

10943 **CHANGE HISTORY**

10944 First released in Issue 1.

10945 **Issue 6**

10946 Extensions beyond the ISO C standard are marked.

10947 **Issue 7**

10948 SD5-XBD-ERN-6 is applied.

## &lt;signal.h&gt;

## 10949 NAME

10950 signal.h — signals

## 10951 SYNOPSIS

10952 #include &lt;signal.h&gt;

## 10953 DESCRIPTION

10954 CX Some of the functionality described on this reference page extends the ISO C standard.  
 10955 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 10956 enable the visibility of these symbols in this header.

10957 The <signal.h> header shall define the following macros, which shall expand to constant  
 10958 expressions with distinct values that have a type compatible with the second argument to, and  
 10959 the return value of, the *signal()* function, and whose values shall compare unequal to the  
 10960 address of any declarable function.

10961 SIG\_DFL Request for default signal handling.

10962 SIG\_ERR Return value from *signal()* in case of error.

10963 OB CX SIG\_HOLD Request that signal be held.

10964 SIG\_IGN Request that signal be ignored.

10965 CX The <signal.h> header shall define the **pthread\_t**, **size\_t**, and **uid\_t** types as described in  
 10966 <sys/types.h>.

10967 The <signal.h> header shall define the **timespec** structure as described in <time.h>.

10968 The <signal.h> header shall define the following data types:

10969 **sig\_atomic\_t** Possibly volatile-qualified integer type of an object that can be accessed as  
 10970 an atomic entity, even in the presence of asynchronous interrupts.

10971 CX **sigset\_t** Integer or structure type of an object used to represent sets of signals.

10972 CX **pid\_t** As described in <sys/types.h>.

10973 CX The <signal.h> header shall define the **pthread\_attr\_t** type as described in <sys/types.h>.

10974 The <signal.h> header shall define the **sigevent** structure, which shall include at least the  
 10975 following members:

10976	int	sigev_notify	Notification type.
10977	int	sigev_signo	Signal number.
10978	union sigval	sigev_value	Signal value.
10979	void	(*sigev_notify_function)(union sigval)	Notification function.
10980			Notification function.
10981	pthread_attr_t	*sigev_notify_attributes	Notification attributes.

10982 The <signal.h> header shall define the following symbolic constants for the values of  
 10983 *sigev\_notify*:

10984 SIGEV\_NONE No asynchronous notification is delivered when the event of interest  
 10985 occurs.

10986 SIGEV\_SIGNAL A queued signal, with an application-defined value, is generated when  
 10987 the event of interest occurs.

10988 SIGEV\_THREAD A notification function is called to perform notification.

10989 The **signal** union shall be defined as:

10990 `int sival_int` Integer signal value.

10991 `void *sival_ptr` Pointer signal value.

10992 The <**signal.h**> header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand to positive integer expressions with type **int**, but which need not be constant expressions. These macros specify a range of signal numbers that are reserved for application use and for which the realtime signal behavior specified in this volume of POSIX.1-2008 is supported. The signal numbers in this range do not overlap any of the signals specified in the following table.

10997 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG\_MAX} signal numbers.

10999 It is implementation-defined whether realtime signal behavior is supported for other signals.

11000 The <**signal.h**> header shall define the following macros that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG followed by an uppercase letter. The macros shall expand to positive integer constant expressions with type **int** and distinct values. The value 0 is reserved for use as the null signal (see *kill()*). Additional implementation-defined signals may occur in the system.

11005 CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, and SIGTERM to be defined.

11007 The following signals shall be supported on all implementations (default actions are explained below the table):

11008

<signal.h>

	Signal	Default Action	Description
11009			
11010	SIGABRT	A	Process abort signal.
11011	SIGALRM	T	Alarm clock.
11012	SIGBUS	A	Access to an undefined portion of a memory object.
11013	SIGCHLD	I	Child process terminated, stopped,
11014	XSI		or continued.
11015	SIGCONT	C	Continue executing, if stopped.
11016	SIGFPE	A	Erroneous arithmetic operation.
11017	SIGHUP	T	Hangup.
11018	SIGILL	A	Illegal instruction.
11019	SIGINT	T	Terminal interrupt signal.
11020	SIGKILL	T	Kill (cannot be caught or ignored).
11021	SIGPIPE	T	Write on a pipe with no one to read it.
11022	SIGQUIT	A	Terminal quit signal.
11023	SIGSEGV	A	Invalid memory reference.
11024	SIGSTOP	S	Stop executing (cannot be caught or ignored).
11025	SIGTERM	T	Termination signal.
11026	SIGTSTP	S	Terminal stop signal.
11027	SIGTTIN	S	Background process attempting read.
11028	SIGTTOU	S	Background process attempting write.
11029	SIGUSR1	T	User-defined signal 1.
11030	SIGUSR2	T	User-defined signal 2.
11031	OB XSR	T	Pollable event.
11032	OB XSR	T	Profiling timer expired.
11033	XSI	A	Bad system call.
11034		A	Trace/breakpoint trap.
11035		I	High bandwidth data is available at a socket.
11036	XSI	T	Virtual timer expired.
11037		A	CPU time limit exceeded.
11038		A	File size limit exceeded.

11039 The default actions are as follows:

- 11040 T Abnormal termination of the process.
- 11041 XSI A Abnormal termination of the process with additional actions.
- 11042 I Ignore the signal.
- 11043 S Stop the process.
- 11044 C Continue the process, if it is stopped; otherwise, ignore the signal.

11045 The effects on the process in each case are described in XSH Section 2.4.3 (on page 486).

11046 CX The <signal.h> header shall declare the **sigaction** structure, which shall include at least the  
11047 following members:

11048	void (*sa_handler) (int)	Pointer to a signal-catching function
11049		or one of the SIG_IGN or SIG_DFL.
11050	sigset_t sa_mask	Set of signals to be blocked during execution
11051		of the signal handling function.
11052	int sa_flags	Special flags.
11053	void (*sa_sigaction) (int, siginfo_t *, void *)	
11054		Pointer to a signal-catching function.

## Headers

## &lt;signal.h&gt;

11055	XSI	The storage occupied by <i>sa_handler</i> and <i>sa_sigaction</i> may overlap, and a conforming application shall not use both simultaneously.	
11056			
11057		The <signal.h> header shall define the following macros which shall expand to integer constant expressions that need not be usable in #if preprocessing directives:	
11058			
11059	CX	<b>SIG_BLOCK</b>	The resulting set is the union of the current set and the signal set pointed to by the argument <i>set</i> .
11060			
11061	CX	<b>SIG_UNBLOCK</b>	The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument <i>set</i> .
11062			
11063	CX	<b>SIG_SETMASK</b>	The resulting set is the signal set pointed to by the argument <i>set</i> .
11064		The <signal.h> header shall also define the following symbolic constants:	
11065	CX	<b>SA_NOCLDSTOP</b>	Do not generate SIGCHLD when children stop or stopped children continue.
11066	XSI		
11067	XSI	<b>SA_ONSTACK</b>	Causes signal delivery to occur on an alternate stack.
11068	CX	<b>SA_RESETHAND</b>	Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.
11069			
11070	CX	<b>SA_RESTART</b>	Causes certain functions to become restartable.
11071	CX	<b>SA_SIGINFO</b>	Causes extra information to be passed to signal handlers at the time of receipt of a signal.
11072			
11073	CX	<b>SA_NOCLDWAIT</b>	Causes implementations not to create zombie processes on child death.
11074	CX	<b>SA_NODEFER</b>	Causes signal not to be automatically blocked on entry to signal handler.
11075	XSI	<b>SS_ONSTACK</b>	Process is executing on an alternate signal stack.
11076	XSI	<b>SS_DISABLE</b>	Alternate signal stack is disabled.
11077	XSI	<b>MINSIGSTKSZ</b>	Minimum stack size for a signal handler.
11078	XSI	<b>SIGSTKSZ</b>	Default size in bytes for the alternate signal stack.
11079	CX	The <signal.h> header shall define the <b>mcontext_t</b> type through <b>typedef</b> .	
11080	CX	The <signal.h> header shall define the <b>ucontext_t</b> type as a structure that shall include at least the following members:	
11081			
11082		<code>ucontext_t *uc_link</code>	Pointer to the context that is resumed when this context returns.
11083			
11084		<code>sigset_t uc_sigmask</code>	The set of signals that are blocked when this context is active.
11085			
11086		<code>stack_t uc_stack</code>	The stack used by this context.
11087		<code>mcontext_t uc_mcontext</code>	A machine-specific representation of the saved context.
11088			
11089		The <signal.h> header shall define the <b>stack_t</b> type as a structure, which shall include at least the following members:	
11090			
11091		<code>void *ss_sp</code>	Stack base or pointer.
11092		<code>size_t ss_size</code>	Stack size.
11093		<code>int ss_flags</code>	Flags.

<signal.h>

11094 CX The <signal.h> header shall define the **siginfo\_t** type as a structure, which shall include at least  
 11095 the following members:

11096	CX	int	si_signo	Signal number.
11097		int	si_code	Signal code.
11098	XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 11099 this signal, as described in <errno.h>.
11100	CX	pid_t	si_pid	Sending process ID.
11101		uid_t	si_uid	Real user ID of sending process.
11102		void	*si_addr	Address of faulting instruction.
11103		int	si_status	Exit value or signal.
11104	OB XSR	long	si_band	Band event for SIGPOLL.
11105	CX	union sigval	si_value	Signal value.

11106 CX The <signal.h> header shall define the symbolic constants in the **Code** column of the following  
 11107 table for use as values of *si\_code* that are signal-specific or non-signal-specific reasons why the  
 11108 signal was generated.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Headers

&lt;signal.h&gt;

	Signal	Code	Reason
11109	SIGILL CX	ILL_ILLOPC	Illegal opcode.
11110		ILL_ILLOPN	Illegal operand.
11111		ILL_ILLADR	Illegal addressing mode.
11112		ILL_ILLTRP	Illegal trap.
11113		ILL_PRVOPC	Privileged opcode.
11114		ILL_PRVREG	Privileged register.
11115		ILL_COPROC	Coprocessor error.
11116		ILL_BADSTK	Internal stack error.
11117	SIGFPE	FPE_INTDIV	Integer divide by zero.
11118		FPE_INTOVF	Integer overflow.
11119		FPE_FLTDIV	Floating-point divide by zero.
11120		FPE_FLTOVF	Floating-point overflow.
11121		FPE_FLTUND	Floating-point underflow.
11122		FPE_FLTRES	Floating-point inexact result.
11123		FPE_FLTINV	Invalid floating-point operation.
11124	SIGSEGV	SEGV_MAPERR	Address not mapped to object.
11125		SEGV_ACCERR	Invalid permissions for mapped object.
11126	SIGBUS	BUS_ADRALN	Invalid address alignment.
11127		BUS_ADRERR	Nonexistent physical address.
11128		BUS_OBJERR	Object-specific hardware error.
11129	SIGTRAP	TRAP_BRKPT	Process breakpoint.
11130		TRAP_TRACE	Process trace trap.
11131	SIGCHLD CX	CLD_EXITED	Child has exited.
11132		CLD_KILLED	Child has terminated abnormally and did not create a <b>core</b> file.
11133		CLD_DUMPED	Child has terminated abnormally and created a <b>core</b> file.
11134		CLD_TRAPPED	Traced child has trapped.
11135		CLD_STOPPED	Child has stopped.
11136		CLD_CONTINUED	Stopped child has continued.
11137	SIGPOLL OB XSR	POLL_IN	Data input available.
11138		POLL_OUT	Output buffers available.
11139		POLL_MSG	Input message available.
11140		POLL_ERR	I/O error.
11141		POLL_PRI	High priority input available.
11142		POLL_HUP	Device disconnected.
11143	Any CX	SI_USER	Signal sent by <i>kill()</i> .
11144		SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
11145		SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
11146		SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
11147		SI_MESGQ	Signal generated by arrival of a message on an empty message queue
11148	CX	Implementations may support additional <i>si_code</i> values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.	
11149			
11150			
11151			

## &lt;signal.h&gt;

11157 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
11158 11159 11160 SIGILL SIGFPE	<b>void</b> * <i>si_addr</i>	Address of faulting instruction.
11161 11162 SIGSEGV SIGBUS	<b>void</b> * <i>si_addr</i>	Address of faulting memory reference.
11163 11164 11165 SIGCHLD	<b>pid_t</b> <i>si_pid</i> <b>int</b> <i>si_status</i> <b>uid_t</b> <i>si_uid</i>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
11166 OB XSR SIGPOLL	<b>long</b> <i>si_band</i>	Band event for POLL_IN, POLL_OUT, or POLL_MSG

11167 For some implementations, the value of *si\_addr* may be inaccurate.

11168 The following shall be declared as functions and may also be defined as macros. Function  
11169 prototypes shall be provided.

```

11170 CX int kill(pid_t, int);
11171 XSI int killpg(pid_t, int);
11172 CX void psiginfo(const siginfo_t *, const char *);
11173 void psignal(int, const char *);
11174 int pthread_kill(pthread_t, int);
11175 int pthread_sigmask(int, const sigset_t *restrict,
11176 sigset_t *restrict);
11177 int raise(int);
11178 CX int sigaction(int, const struct sigaction *restrict,
11179 struct sigaction *restrict);
11180 int sigaddset(sigset_t *, int);
11181 XSI int sigaltstack(const stack_t *restrict, stack_t *restrict);
11182 CX int sigdelset(sigset_t *, int);
11183 int sigemptyset(sigset_t *);
11184 int sigfillset(sigset_t *);
11185 OB XSI int sighold(int);
11186 int sigignore(int);
11187 int siginterrupt(int, int);
11188 CX int sigismember(const sigset_t *, int);
11189 void (*signal(int, void (*)(int)))(int);
11190 OB XSI int sigpause(int);
11191 CX int sigpending(sigset_t *);
11192 int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11193 int sigqueue(pid_t, int, const union sigval);
11194 OB XSI int sigrelse(int);
11195 void (*sigset(int, void (*)(int)))(int);
11196 CX int sigsuspend(const sigset_t *);
11197 int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11198 const struct timespec *restrict);
11199 int sigwait(const sigset_t *restrict, int *restrict);
11200 int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);

```

11201 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

11202 **APPLICATION USAGE**

11203 On systems not supporting the XSI option, the *si\_pid* and *si\_uid* members of **siginfo\_t** are only  
 11204 required to be valid when *si\_code* is SI\_USER or SI\_QUEUE. On XSI-conforming systems, they  
 11205 are also valid for all *si\_code* values less than or equal to 0; however, it is unspecified whether  
 11206 SI\_USER and SI\_QUEUE have values less than or equal to zero, and therefore XSI applications  
 11207 should check whether *si\_code* has the value SI\_USER or SI\_QUEUE or is less than or equal to 0 to  
 11208 tell whether *si\_pid* and *si\_uid* are valid.

11209 **RATIONALE**

11210 None.

11211 **FUTURE DIRECTIONS**

11212 The SIGPOLL and SIGPROF signals may be removed in a future version.

11213 **SEE ALSO**

11214 <errno.h>, <stropts.h>, <sys/types.h>, <time.h>

11215 XSH Section 2.2 (on page 468), *alarm()*, *ioctl()*, *kill()*, *killpg()*, *psiginfo()*, *pthread\_kill()*,  
 11216 *pthread\_sigmask()*, *raise()*, *sigaction()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*,  
 11217 *sigfillset()*, *sighold()*, *siginterrupt()*, *sigismember()*, *signal()*, *sigpending()*, *sigqueue()*, *sigsuspend()*,  
 11218 *sigtimedwait()*, *sigwait()*, *timer\_create()*, *wait()*, *waitid()*

11219 **CHANGE HISTORY**

11220 First released in Issue 1.

11221 **Issue 5**

11222 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 11223 Threads Extension.

11224 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is  
 11225 removed.

11226 **Issue 6**

11227 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for  
 11228 abnormal termination is clarified.

11229 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*  
 11230 function.

11231 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev\_notify\_function*  
 11232 function member of the **sigevent** structure.

11233 The following new requirements on POSIX implementations derive from alignment with the  
 11234 Single UNIX Specification:

- 11235 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now  
 11236 mandated. This is also a FIPS requirement.
- 11237 • The **pid\_t** definition is mandated.

11238 The RT markings are changed to RTS to denote that the semantics are part of the Realtime  
 11239 Signals Extension option.

11240 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,  
 11241 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

11242 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from  
 11243 <time.h> may be made visible when <signal.h> is included.

11244 Extensions beyond the ISO C standard are marked.

- 11245 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive  
11246 text for members of the **sigaction** structure.
- 11247 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of  
11248 the *sa\_sigaction* member of the **sigaction** structure.
- 11249 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of  
11250 the **siginfo\_t** type structure in the DESCRIPTION. This is an editorial change and no normative  
11251 change is intended.
- 11252 **Issue 7**
- 11253 SD5-XBD-ERN-5 is applied.
- 11254 SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed  
11255 at the same time as the LEGACY *sigstack()* function.
- 11256 SD5-XBD-ERN-56 is applied, adding a reference to <**sys/types.h**> for the **size\_t** type.
- 11257 Austin Group Interpretation 1003.1-2001 #034 is applied.
- 11258 The **ucontext\_t** and **mcontext\_t** structures are added here from the obsolescent <**ucontext.h**>  
11259 header.
- 11260 The *psiginfo()* and *psignal()* functions are added from The Open Group Technical Standard, 2006,  
11261 Extended API Set Part 1.
- 11262 The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked  
11263 obsolescent.
- 11264 The SA\_RESETHAND, SA\_RESTART, SA\_SIGINFO, SA\_NOCLDWAIT, and SA\_NODEFER  
11265 constants are moved from the XSI option to the Base.
- 11266 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 11267 This reference page is clarified with respect to macros and symbolic constants, and declarations  
11268 for the **pthread\_attr\_t**, **pthread\_t**, and **uid\_t** types and the **timespec** structure are added.
- 11269 SIGRTMIN and SIGRTMAX are required to be positive integer expressions.
- 11270 The APPLICATION USAGE section is updated to describe the *si\_pid* and *si\_uid* members of  
11271 **siginfo\_t**.

## 11272 NAME

11273 spawn.h — spawn (ADVANCED REALTIME)

## 11274 SYNOPSIS

11275 SPN #include &lt;spawn.h&gt;

## 11276 DESCRIPTION

11277 The <spawn.h> header shall define the **posix\_spawnattr\_t** and **posix\_spawn\_file\_actions\_t**  
11278 types used in performing spawn operations.11279 The <spawn.h> header shall define the **mode\_t** and **pid\_t** types as described in <sys/types.h>.11280 The <spawn.h> header shall define the **sigset\_t** type as described in <signal.h>.11281 The tag **sched\_param** shall be declared as naming an incomplete structure type, the contents of  
11282 which are described in the <sched.h> header.11283 The <spawn.h> header shall define the following symbolic constants for use as the flags that  
11284 may be set in a **posix\_spawnattr\_t** object using the *posix\_spawnattr\_setflags()* function:

11285 POSIX\_SPAWN\_RESETIDS

11286 POSIX\_SPAWN\_SETPGROUP

11287 PS POSIX\_SPAWN\_SETSCHEDPARAM

11288 POSIX\_SPAWN\_SETSCHEDULER

11289 POSIX\_SPAWN\_SETSIGDEF

11290 POSIX\_SPAWN\_SETSIGMASK

11291 The following shall be declared as functions and may also be defined as macros. Function  
11292 prototypes shall be provided.

```

11293 int  posix_spawn(pid_t *restrict, const char *restrict,
11294                const posix_spawn_file_actions_t *,
11295                const posix_spawnattr_t *restrict, char *const [restrict],
11296                char *const [restrict]);
11297 int  posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11298                int);
11299 int  posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11300                int, int);
11301 int  posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11302                int, const char *restrict, int, mode_t);
11303 int  posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11304 int  posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11305 int  posix_spawnattr_destroy(posix_spawnattr_t *);
11306 int  posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11307                short *restrict);
11308 int  posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11309                pid_t *restrict);
11310 PS  int  posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11311                struct sched_param *restrict);
11312 int  posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11313                int *restrict);
11314 int  posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11315                sigset_t *restrict);
11316 int  posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11317                sigset_t *restrict);
11318 int  posix_spawnattr_init(posix_spawnattr_t *);

```

## &lt;spawn.h&gt;

Headers

```

11319     int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11320     int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
11321 PS    int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11322         const struct sched_param *restrict);
11323     int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11324     int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11325         const sigset_t *restrict);
11326     int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11327         const sigset_t *restrict);
11328     int    posix_spawn(pid_t *restrict, const char *restrict,
11329         const posix_spawn_file_actions_t *,
11330         const posix_spawnattr_t *restrict,
11331         char *const [restrict], char *const [restrict]);

```

11332 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h> and  
 11333 <signal.h> headers.

## 11334 APPLICATION USAGE

11335 None.

## 11336 RATIONALE

11337 None.

## 11338 FUTURE DIRECTIONS

11339 None.

## 11340 SEE ALSO

11341 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>

11342 XSH *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
 11343 *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getflags()*,  
 11344 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
 11345 *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getsigmask()*

## 11346 CHANGE HISTORY

11347 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11348 The **restrict** keyword is added to the prototypes for *posix\_spawn()*,  
 11349 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
 11350 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
 11351 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setschedparam()*,  
 11352 *posix\_spawnattr\_setsigmask()*, and *posix\_spawn()*.

## 11353 Issue 7

11354 This reference page is clarified with respect to macros and symbolic constants, and declarations  
 11355 for the **mode\_t**, **pid\_t**, and **sigset\_t** types are added.

## 11356 NAME

11357        stdarg.h — handle variable argument list

## 11358 SYNOPSIS

```
11359        #include <stdarg.h>

11360        void va_start(va_list ap, argN);
11361        void va_copy(va_list dest, va_list src);
11362        type va_arg(va_list ap, type);
11363        void va_end(va_list ap);
```

## 11364 DESCRIPTION

11365 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 11366 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11367 volume of POSIX.1-2008 defers to the ISO C standard.

11368       The <stdarg.h> header shall contain a set of macros which allows portable functions that accept  
 11369 variable argument lists to be written. Functions that have variable argument lists (such as  
 11370 *printf()*) but do not use these macros are inherently non-portable, as different systems use  
 11371 different argument-passing conventions.

11372       The <stdarg.h> header shall define the **va\_list** type for variables used to traverse the list.

11373       The *va\_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to  
 11374 *va\_arg()*.

11375       The *va\_copy()* macro initializes *dest* as a copy of *src*, as if the *va\_start()* macro had been applied  
 11376 to *dest* followed by the same sequence of uses of the *va\_arg()* macro as had previously been used  
 11377 to reach the present state of *src*. Neither the *va\_copy()* nor *va\_start()* macro shall be invoked to  
 11378 reinitialize *dest* without an intervening invocation of the *va\_end()* macro for the same *dest*.

11379       The object *ap* may be passed as an argument to another function; if that function invokes the  
 11380 *va\_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall  
 11381 be passed to the *va\_end()* macro prior to any further reference to *ap*. The parameter *argN* is the  
 11382 identifier of the rightmost parameter in the variable parameter list in the function definition (the  
 11383 one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a  
 11384 function type or array type, or with a type that is not compatible with the type that results after  
 11385 application of the default argument promotions, the behavior is undefined.

11386       The *va\_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation  
 11387 of *va\_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*  
 11388 parameter shall be a type name specified such that the type of a pointer to an object that has the  
 11389 specified type can be obtained simply by postfixing a '\*' to type. If there is no actual next  
 11390 argument, or if *type* is not compatible with the type of the actual next argument (as promoted  
 11391 according to the default argument promotions), the behavior is undefined, except for the  
 11392 following cases:

- 11393       • One type is a signed integer type, the other type is the corresponding unsigned integer  
 11394 type, and the value is representable in both types.
- 11395       • One type is a pointer to **void** and the other is a pointer to a character type.
- 11396 XSI     • Both types are pointers.

11397       Different types can be mixed, but it is up to the routine to know what type of argument is  
 11398 expected.

11399       The *va\_end()* macro is used to clean up; it invalidates *ap* for use (unless *va\_start()* or *va\_copy()* is  
 11400 invoked again).

## &lt;stdarg.h&gt;

11401 Each invocation of the *va\_start()* and *va\_copy()* macros shall be matched by a corresponding  
 11402 invocation of the *va\_end()* macro in the same function.

11403 Multiple traversals, each bracketed by *va\_start()* ... *va\_end()*, are possible.

11404 **EXAMPLES**

11405 This example is a possible implementation of *execl()*:

```
11406 #include <stdarg.h>
11407 #define MAXARGS 31
11408 /*
11409  * execl is called by
11410  * execl(file, arg1, arg2, ..., (char *) (0));
11411  */
11412 int execl(const char *file, const char *args, ...)
11413 {
11414     va_list ap;
11415     char *array[MAXARGS + 1];
11416     int argno = 0;
11417     va_start(ap, args);
11418     while (args != 0 && argno < MAXARGS)
11419     {
11420         array[argno++] = args;
11421         args = va_arg(ap, const char *);
11422     }
11423     array[argno] = (char *) 0;
11424     va_end(ap);
11425     return execv(file, array);
11426 }
```

11427 **APPLICATION USAGE**

11428 It is up to the calling routine to communicate to the called routine how many arguments there  
 11429 are, since it is not always possible for the called routine to determine this in any other way. For  
 11430 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell  
 11431 how many arguments are there by the *format* argument.

11432 **RATIONALE**

11433 None.

11434 **FUTURE DIRECTIONS**

11435 None.

11436 **SEE ALSO**

11437 XSH *exec*, *fprintf()*

11438 **CHANGE HISTORY**

11439 First released in Issue 4. Derived from the ANSI C standard.

11440 **Issue 6**

11441 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11442 **NAME**

11443            stdbool.h — boolean type and values

11444 **SYNOPSIS**

11445            #include &lt;stdbool.h&gt;

11446 **DESCRIPTION**

11447 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
 11448            conflict between the requirements described here and the ISO C standard is unintentional. This  
 11449            volume of POSIX.1-2008 defers to the ISO C standard.

11450            The &lt;stdbool.h&gt; header shall define the following macros:

11451            bool     Expands to **\_Bool**.

11452            true     Expands to the integer constant 1.

11453            false    Expands to the integer constant 0.

11454            \_\_bool\_true\_false\_are\_defined

11455            Expands to the integer constant 1.

11456            An application may undefine and then possibly redefine the macros bool, true, and false.

11457 **APPLICATION USAGE**

11458            None.

11459 **RATIONALE**

11460            None.

11461 **FUTURE DIRECTIONS**

11462            The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature  
 11463            and may be removed in a future version.

11464 **SEE ALSO**

11465            None.

11466 **CHANGE HISTORY**

11467            First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

**<stddef.h>**

Headers

11468 **NAME**11469           **stddef.h** — standard type definitions11470 **SYNOPSIS**

11471           #include &lt;stddef.h&gt;

11472 **DESCRIPTION**

11473 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 11474           conflict between the requirements described here and the ISO C standard is unintentional. This  
 11475           volume of POSIX.1-2008 defers to the ISO C standard.

11476       The **<stddef.h>** header shall define the following macros:

11477 CX       **NULL**       Null pointer constant. The macro shall expand to an integer constant expression  
 11478           with the value 0 cast to type **void \***.

11479       **offsetof**(*type*, *member-designator*)

11480           Integer constant expression of type **size\_t**, the value of which is the offset in bytes  
 11481           to the structure member (*member-designator*), from the beginning of its structure  
 11482           (*type*).

11483       The **<stddef.h>** header shall define the following types:11484       **ptrdiff\_t**   Signed integer type of the result of subtracting two pointers.

11485       **wchar\_t**     Integer type whose range of values can represent distinct codes for all members of  
 11486           the largest extended character set specified among the supported locales; the null  
 11487           character shall have the code value zero. Each member of the basic character set  
 11488           shall have a code value equal to its value when used as the lone character in an  
 11489           integer character constant if an implementation does not define  
 11490           \_\_STDC\_MB\_MIGHT\_NEQ\_WC\_\_.

11491       **size\_t**       Unsigned integer type of the result of the *sizeof* operator.

11492       The implementation shall support one or more programming environments in which the widths  
 11493       of **ptrdiff\_t**, **size\_t**, and **wchar\_t** are no greater than the width of type **long**. The names of these  
 11494       programming environments can be obtained using the *confstr*() function or the *getconf* utility.

11495 **APPLICATION USAGE**

11496       None.

11497 **RATIONALE**

11498       The ISO C standard does not require the **NULL** macro to include the cast to type **void \*** and  
 11499       specifies that the **NULL** macro be implementation-defined. POSIX.1-2008 requires the cast and  
 11500       therefore need not be implementation-defined.

11501 **FUTURE DIRECTIONS**

11502       None.

11503 **SEE ALSO**11504       [<sys/types.h>](#), [<wchar.h>](#)11505       XSH *confstr*()11506       XCU *getconf*11507 **CHANGE HISTORY**

11508       First released in Issue 4. Derived from the ANSI C standard.

*Headers*

&lt;stddef.h&gt;

11509 **Issue 7**

11510 This reference page is clarified with respect to macros and symbolic constants.

11511 SD5-XBD-ERN-53 is applied, updating the definition of `wchar_t` to align with  
11512 ISO/IEC 9899:1999 standard, Technical Corrigendum 3.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<stdint.h>**

Headers

11513 **NAME**

11514           stdint.h — integer types

11515 **SYNOPSIS**

11516           #include &lt;stdint.h&gt;

11517 **DESCRIPTION**

11518 **CX**       Some of the functionality described on this reference page extends the ISO C standard.  
 11519       Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 11520       enable the visibility of these symbols in this header.

11521       The **<stdint.h>** header shall declare sets of integer types having specified widths, and shall  
 11522       define corresponding sets of macros. It shall also define macros that specify limits of integer  
 11523       types corresponding to types defined in other standard headers.

11524       **Note:**     The “width” of an integer type is the number of bits used to store its value in a pure binary  
 11525       system; the actual type may use more bits than that (for example, a 28-bit type could be stored  
 11526       in 32 bits of actual storage). An  $N$ -bit signed type has values in the range  $-2^{N-1}$  or  $1-2^{N-1}$  to  
 11527        $2^{N-1}-1$ , while an  $N$ -bit unsigned type has values in the range 0 to  $2^N-1$ .

11528       Types are defined in the following categories:

- 11529           • Integer types having certain exact widths
- 11530           • Integer types having at least certain specified widths
- 11531           • Fastest integer types having at least certain specified widths
- 11532           • Integer types wide enough to hold pointers to objects
- 11533           • Integer types having greatest width

11534       (Some of these types may denote the same type.)

11535       Corresponding macros specify limits of the declared types and construct suitable constants.

11536       For each type described herein that the implementation provides, the **<stdint.h>** header shall  
 11537       declare that **typedef** name and define the associated macros. Conversely, for each type described  
 11538       herein that the implementation does not provide, the **<stdint.h>** header shall not declare that  
 11539       **typedef** name, nor shall it define the associated macros. An implementation shall provide those  
 11540       types described as required, but need not provide any of the others (described as optional).

11541 **Integer Types**

11542       When **typedef** names differing only in the absence or presence of the initial  $u$  are defined, they  
 11543       shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999  
 11544       standard, Section 6.2.5; an implementation providing one of these corresponding types shall also  
 11545       provide the other.

11546       In the following descriptions, the symbol  $N$  represents an unsigned decimal integer with no  
 11547       leading zeros (for example, 8 or 24, but not 04 or 048).

- 11548           • Exact-width integer types

11549       The **typedef** name **int $N$ \_t** designates a signed integer type with width  $N$ , no padding bits,  
 11550       and a two’s-complement representation. Thus, **int8\_t** denotes a signed integer type with a  
 11551       width of exactly 8 bits.

11552       The **typedef** name **uint $N$ \_t** designates an unsigned integer type with width  $N$ . Thus,  
 11553       **uint24\_t** denotes an unsigned integer type with a width of exactly 24 bits.

11554	CX	The following types are required:
11555		<b>int8_t</b>
11556		<b>int16_t</b>
11557		<b>int32_t</b>
11558		<b>uint8_t</b>
11559		<b>uint16_t</b>
11560		<b>uint32_t</b>
11561		If an implementation provides integer types with width 64 that meet these requirements,
11562		then the following types are required:
11563		<b>int64_t</b>
11564		<b>uint64_t</b>
11565	CX	In particular, this will be the case if any of the following are true:
11566		— The implementation supports the <code>_POSIX_V7_ILP32_OFFBIG</code> programming
11567		environment and the application is being built in the <code>_POSIX_V7_ILP32_OFFBIG</code>
11568		programming environment (see the Shell and Utilities volume of POSIX.1-2008, <i>c99</i> ,
11569		Programming Environments).
11570		— The implementation supports the <code>_POSIX_V7_LP64_OFF64</code> programming
11571		environment and the application is being built in the <code>_POSIX_V7_LP64_OFF64</code>
11572		programming environment.
11573		— The implementation supports the <code>_POSIX_V7_LPBIG_OFFBIG</code> programming
11574		environment and the application is being built in the <code>_POSIX_V7_LPBIG_OFFBIG</code>
11575		programming environment.
11576		All other types of this form are optional.
11577		• Minimum-width integer types
11578		The <b>typedef</b> name <b>int_leastN_t</b> designates a signed integer type with a width of at least <i>N</i> ,
11579		such that no signed integer type with lesser size has at least the specified width. Thus,
11580		<b>int_least32_t</b> denotes a signed integer type with a width of at least 32 bits.
11581		The <b>typedef</b> name <b>uint_leastN_t</b> designates an unsigned integer type with a width of at
11582		least <i>N</i> , such that no unsigned integer type with lesser size has at least the specified width.
11583		Thus, <b>uint_least16_t</b> denotes an unsigned integer type with a width of at least 16 bits.
11584		The following types are required:
11585		<b>int_least8_t</b>
11586		<b>int_least16_t</b>
11587		<b>int_least32_t</b>
11588		<b>int_least64_t</b>
11589		<b>uint_least8_t</b>
11590		<b>uint_least16_t</b>
11591		<b>uint_least32_t</b>
11592		<b>uint_least64_t</b>
11593		All other types of this form are optional.

- 11594
- Fastest minimum-width integer types
- 11595 Each of the following types designates an integer type that is usually fastest to operate  
11596 with among all integer types that have at least the specified width.
- 11597 The designated type is not guaranteed to be fastest for all purposes; if the implementation  
11598 has no clear grounds for choosing one type over another, it will simply pick some integer  
11599 type satisfying the signedness and width requirements.
- 11600 The **typedef** name **int\_fastN\_t** designates the fastest signed integer type with a width of at  
11601 least *N*. The **typedef** name **uint\_fastN\_t** designates the fastest unsigned integer type with  
11602 a width of at least *N*.
- 11603 The following types are required:
- 11604 **int\_fast8\_t**  
11605 **int\_fast16\_t**  
11606 **int\_fast32\_t**  
11607 **int\_fast64\_t**  
11608 **uint\_fast8\_t**  
11609 **uint\_fast16\_t**  
11610 **uint\_fast32\_t**  
11611 **uint\_fast64\_t**
- 11612 All other types of this form are optional.
- Integer types capable of holding object pointers
- 11614 The following type designates a signed integer type with the property that any valid  
11615 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
11616 the result will compare equal to the original pointer:
- 11617 **intptr\_t**
- 11618 The following type designates an unsigned integer type with the property that any valid  
11619 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
11620 the result will compare equal to the original pointer:
- 11621 **uintptr\_t**
- 11622 XSI On XSI-conformant systems, the **intptr\_t** and **uintptr\_t** types are required; otherwise, they  
11623 are optional.
- Greatest-width integer types
- 11625 The following type designates a signed integer type capable of representing any value of  
11626 any signed integer type:
- 11627 **intmax\_t**
- 11628 The following type designates an unsigned integer type capable of representing any value  
11629 of any unsigned integer type:
- 11630 **uintmax\_t**
- 11631 These types are required.

11632 **Note:** Applications can test for optional types by using the corresponding limit macro from [Limits of](#)  
11633 [Specified-Width Integer Types](#).

### 11634 **Limits of Specified-Width Integer Types**

11635 The following macros specify the minimum and maximum limits of the types declared in the  
11636 <stdint.h> header. Each macro name corresponds to a similar type name in [Integer Types](#) (on  
11637 page 344).

11638 Each instance of any defined macro shall be replaced by a constant expression suitable for use in  
11639 #if preprocessing directives, and this expression shall have the same type as would an  
11640 expression that is an object of the corresponding type converted according to the integer  
11641 promotions. Its implementation-defined value shall be equal to or greater in magnitude  
11642 (absolute value) than the corresponding value given below, with the same sign, except where  
11643 stated to be exactly the given value.

- 11644 • Limits of exact-width integer types

- 11645 — Minimum values of exact-width signed integer types:

11646 {INTN\_MIN} Exactly  $-(2^{N-1})$

- 11647 — Maximum values of exact-width signed integer types:

11648 {INTN\_MAX} Exactly  $2^{N-1} - 1$

- 11649 — Maximum values of exact-width unsigned integer types:

11650 {UINTN\_MAX} Exactly  $2^N - 1$

- 11651 • Limits of minimum-width integer types

- 11652 — Minimum values of minimum-width signed integer types:

11653 {INT\_LEASTN\_MIN}  $-(2^{N-1} - 1)$

- 11654 — Maximum values of minimum-width signed integer types:

11655 {INT\_LEASTN\_MAX}  $2^{N-1} - 1$

- 11656 — Maximum values of minimum-width unsigned integer types:

11657 {UINT\_LEASTN\_MAX}  $2^N - 1$

- 11658 • Limits of fastest minimum-width integer types

- 11659 — Minimum values of fastest minimum-width signed integer types:

11660 {INT\_FASTN\_MIN}  $-(2^{N-1} - 1)$

- 11661 — Maximum values of fastest minimum-width signed integer types:

11662 {INT\_FASTN\_MAX}  $2^{N-1} - 1$

- 11663 — Maximum values of fastest minimum-width unsigned integer types:

11664 {UINT\_FASTN\_MAX}  $2^N - 1$

- 11665 • Limits of integer types capable of holding object pointers

- 11666 — Minimum value of pointer-holding signed integer type:

11667 {INTPTR\_MIN}  $-(2^{15} - 1)$

## &lt;stdint.h&gt;

- 11668 — Maximum value of pointer-holding signed integer type:  
 11669 {INTPTR\_MAX}  $2^{15} - 1$
- 11670 — Maximum value of pointer-holding unsigned integer type:  
 11671 {UINTPTR\_MAX}  $2^{16} - 1$
- 11672 • Limits of greatest-width integer types
    - 11673 — Minimum value of greatest-width signed integer type:  
 11674 {INTMAX\_MIN}  $-(2^{63} - 1)$
    - 11675 — Maximum value of greatest-width signed integer type:  
 11676 {INTMAX\_MAX}  $2^{63} - 1$
    - 11677 — Maximum value of greatest-width unsigned integer type:  
 11678 {UINTMAX\_MAX}  $2^{64} - 1$

**Limits of Other Integer Types**

11680 The following macros specify the minimum and maximum limits of integer types corresponding  
 11681 to types defined in other standard headers.

11682 Each instance of these macros shall be replaced by a constant expression suitable for use in #if  
 11683 preprocessing directives, and this expression shall have the same type as would an expression  
 11684 that is an object of the corresponding type converted according to the integer promotions. Its  
 11685 implementation-defined value shall be equal to or greater in magnitude (absolute value) than  
 11686 the corresponding value given below, with the same sign.

- 11687 • Limits of **ptrdiff\_t**:
  - 11688 {PTRDIFF\_MIN}  $-65\,535$
  - 11689 {PTRDIFF\_MAX}  $+65\,535$
- 11690 • Limits of **sig\_atomic\_t**:
  - 11691 {SIG\_ATOMIC\_MIN} See below.
  - 11692 {SIG\_ATOMIC\_MAX} See below.
- 11693 • Limit of **size\_t**:
  - 11694 {SIZE\_MAX}  $65\,535$
- 11695 • Limits of **wchar\_t**:
  - 11696 {WCHAR\_MIN} See below.
  - 11697 {WCHAR\_MAX} See below.
- 11698 • Limits of **wint\_t**:
  - 11699 {WINT\_MIN} See below.
  - 11700 {WINT\_MAX} See below.

11701 If **sig\_atomic\_t** (see the <signal.h> header) is defined as a signed integer type, the value of  
 11702 {SIG\_ATOMIC\_MIN} shall be no greater than  $-127$  and the value of {SIG\_ATOMIC\_MAX} shall  
 11703 be no less than  $127$ ; otherwise, **sig\_atomic\_t** shall be defined as an unsigned integer type, and  
 11704 the value of {SIG\_ATOMIC\_MIN} shall be  $0$  and the value of {SIG\_ATOMIC\_MAX} shall be no

11705 less than 255.

11706 If **wchar\_t** (see the <stddef.h> header) is defined as a signed integer type, the value of  
 11707 {WCHAR\_MIN} shall be no greater than -127 and the value of {WCHAR\_MAX} shall be no less  
 11708 than 127; otherwise, **wchar\_t** shall be defined as an unsigned integer type, and the value of  
 11709 {WCHAR\_MIN} shall be 0 and the value of {WCHAR\_MAX} shall be no less than 255.

11710 If **wint\_t** (see the <wchar.h> header) is defined as a signed integer type, the value of  
 11711 {WINT\_MIN} shall be no greater than -32767 and the value of {WINT\_MAX} shall be no less  
 11712 than 32767; otherwise, **wint\_t** shall be defined as an unsigned integer type, and the value of  
 11713 {WINT\_MIN} shall be 0 and the value of {WINT\_MAX} shall be no less than 65535.

#### 11714 **Macros for Integer Constant Expressions**

11715 The following macros expand to integer constant expressions suitable for initializing objects that  
 11716 have integer types corresponding to types defined in the <stdint.h> header. Each macro name  
 11717 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*  
 11718 *integer types*.

11719 Each invocation of one of these macros shall expand to an integer constant expression suitable  
 11720 for use in **#if** preprocessing directives. The type of the expression shall have the same type as  
 11721 would an expression that is an object of the corresponding type converted according to the  
 11722 integer promotions. The value of the expression shall be that of the argument.

11723 The argument in any instance of these macros shall be an unsuffixed integer constant with a  
 11724 value that does not exceed the limits for the corresponding type.

- 11725 • Macros for minimum-width integer constant expressions

11726 The macro *INTN\_C(value)* shall expand to an integer constant expression corresponding to  
 11727 the type **int\_leastN\_t**. The macro *UINTN\_C(value)* shall expand to an integer constant  
 11728 expression corresponding to the type **uint\_leastN\_t**. For example, if **uint\_least64\_t** is a  
 11729 name for the type **unsigned long long**, then *UINT64\_C(0x123)* might expand to the integer  
 11730 constant 0x123ULL.

- 11731 • Macros for greatest-width integer constant expressions

11732 The following macro expands to an integer constant expression having the value specified  
 11733 by its argument and the type **intmax\_t**:

11734 *INTMAX\_C(value)*

11735 The following macro expands to an integer constant expression having the value specified  
 11736 by its argument and the type **uintmax\_t**:

11737 *UINTMAX\_C(value)*

#### 11738 **APPLICATION USAGE**

11739 None.

#### 11740 **RATIONALE**

11741 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in  
 11742 freestanding environments, which might not support the formatted I/O functions. In some  
 11743 environments, if the formatted conversion support is not wanted, using this header instead of  
 11744 the <inttypes.h> header avoids defining such a large number of macros.

11745 As a consequence of adding `int8_t`, the following are true:

11746

- A byte is exactly 8 bits.
- {CHAR\_BIT} has the value 8, {SCHAR\_MAX} has the value 127, {SCHAR\_MIN} has the value -128, and {UCHAR\_MAX} has the value 255.

11747

11748

11749 (The POSIX standard explicitly requires 8-bit char and two's-complement arithmetic.)

11750 **FUTURE DIRECTIONS**

11751 `typedef` names beginning with `int` or `uint` and ending with `_t` may be added to the types defined

11752 in the `<stdint.h>` header. Macro names beginning with `INT` or `UINT` and ending with `_MAX`,

11753 `_MIN`, or `_C` may be added to the macros defined in the `<stdint.h>` header.

11754 **SEE ALSO**

11755 [<inttypes.h>](#), [<signal.h>](#), [<stddef.h>](#), [<wchar.h>](#)

11756 XSH Section 2.2 (on page 468)

11757 **CHANGE HISTORY**

11758 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11759 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is applied.

11760 **Issue 7**

11761 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied.

11762 SD5-XBD-ERN-67 is applied.

## Headers

## &lt;stdio.h&gt;

## 11763 NAME

11764       stdio.h — standard buffered input/output

## 11765 SYNOPSIS

11766       #include &lt;stdio.h&gt;

## 11767 DESCRIPTION

11768 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 11769       Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 11770       enable the visibility of these symbols in this header.

11771       The <stdio.h> header shall define the following data types through **typedef**:11772       **FILE**                A structure containing information about a file.11773       **fpos\_t**               A non-array type containing all information needed to specify uniquely  
 11774       every position within a file.11775       **off\_t**                As described in <sys/types.h>.11776       **size\_t**               As described in <stddef.h>.11777 CX       **ssize\_t**            As described in <sys/types.h>.11778 CX       **va\_list**            As described in <stdarg.h>.11779       The <stdio.h> header shall define the following macros which shall expand to integer constant  
 11780       expressions:11781 CX       **BUFSIZ**            Size of <stdio.h> buffers. This shall expand to a positive value.11782 CX       **L\_ctermid**           Maximum size of character array to hold *ctermid()* output.11783 OB       **L\_tmpnam**            Maximum size of character array to hold *tmpnam()* output.11784       The <stdio.h> header shall define the following macros which shall expand to integer constant  
 11785       expressions with distinct values:11786       **\_IOFBF**            Input/output fully buffered.11787       **\_IOLBF**            Input/output line buffered.11788       **\_IONBF**            Input/output unbuffered.11789       The <stdio.h> header shall define the following macros which shall expand to integer constant  
 11790       expressions with distinct values:11791       **SEEK\_CUR**            Seek relative to current position.11792       **SEEK\_END**            Seek relative to end-of-file.11793       **SEEK\_SET**            Seek relative to start-of-file.11794       The <stdio.h> header shall define the following macros which shall expand to integer constant  
 11795       expressions denoting implementation limits:11796       {FILENAME\_MAX}       Maximum size in bytes of the longest filename string that the  
 11797       implementation guarantees can be opened.11798       {FOPEN\_MAX}         Number of streams which the implementation guarantees can be open  
 11799       simultaneously. The value is at least eight.11800 OB       {TMP\_MAX}           Minimum number of unique filenames generated by *tmpnam()*.  
 11801       Maximum number of times an application can call *tmpnam()* reliably. The  
 11802       value of {TMP\_MAX} is at least 25.

## &lt;stdio.h&gt;

11803	OB XSI	On XSI-conformant systems, the value of {TMP_MAX} is at least 10 000.
11804		The <stdio.h> header shall define the following macro which shall expand to an integer constant expression with type <b>int</b> and a negative value:
11805		
11806		EOF                      End-of-file return value.
11807		The <stdio.h> header shall define NULL as described in <stddef.h>.
11808		The <stdio.h> header shall define the following macro which shall expand to a string constant:
11809	OB XSI	P_tmpdir                Default directory prefix for <i>tempnam()</i> .
11810		The <stdio.h> header shall define the following macros which shall expand to expressions of type "pointer to <b>FILE</b> " that point to the <b>FILE</b> objects associated, respectively, with the standard error, input, and output streams:
11811		
11812		
11813		<i>stderr</i> Standard error output stream.
11814		<i>stdin</i> Standard input stream.
11815		<i>stdout</i> Standard output stream.
11816		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
11817		
11818		void      clearerr(FILE *);
11819	CX	char     *ctermid(char *);
11820		int      dprintf(int, const char *restrict, ...)
11821		int      fclose(FILE *);
11822	CX	FILE    *fdopen(int, const char *);
11823		int      feof(FILE *);
11824		int      ferror(FILE *);
11825		int      fflush(FILE *);
11826		int      fgetc(FILE *);
11827		int      fgetpos(FILE *restrict, fpos_t *restrict);
11828		char    *fgets(char *restrict, int, FILE *restrict);
11829	CX	int      fileno(FILE *);
11830		void     flockfile(FILE *);
11831		FILE    *fmemopen(void *restrict, size_t, const char *restrict);
11832		FILE    *fopen(const char *restrict, const char *restrict);
11833		int      fprintf(FILE *restrict, const char *restrict, ...);
11834		int      fputc(int, FILE *);
11835		int      fputs(const char *restrict, FILE *restrict);
11836		size_t   fread(void *restrict, size_t, size_t, FILE *restrict);
11837		FILE    *freopen(const char *restrict, const char *restrict, FILE *restrict);
11838		
11839		int      fscanf(FILE *restrict, const char *restrict, ...);
11840		int      fseek(FILE *, long, int);
11841	CX	int      fseeko(FILE *, off_t, int);
11842		int      fsetpos(FILE *, const fpos_t *);
11843		long     ftell(FILE *);
11844	CX	off_t    ftello(FILE *);
11845		int      ftrylockfile(FILE *);
11846		void     funlockfile(FILE *);
11847		size_t   fwrite(const void *restrict, size_t, size_t, FILE *restrict);
11848		int      getc(FILE *);

## Headers

## &lt;stdio.h&gt;

```

11849      int      getchar(void);
11850 CX      int      getc_unlocked(FILE *);
11851      int      getchar_unlocked(void);
11852      ssize_t  getdelim(char **restrict, size_t *restrict, int,
11853                  FILE *restrict);
11854      ssize_t  getline(char **restrict, size_t *restrict, FILE *restrict);
11855 OB      char      *gets(char *);
11856 CX      FILE      *open_memstream(char **, size_t *);
11857      int      pclose(FILE *);
11858      void     perror(const char *);
11859 CX      FILE      *popen(const char *, const char *);
11860      int      printf(const char *restrict, ...);
11861      int      putc(int, FILE *);
11862      int      putchar(int);
11863 CX      int      putc_unlocked(int, FILE *);
11864      int      putchar_unlocked(int);
11865      int      puts(const char *);
11866      int      remove(const char *);
11867      int      rename(const char *, const char *);
11868 CX      int      renameat(int, const char *, int, const char *);
11869      void     rewind(FILE *);
11870      int      scanf(const char *restrict, ...);
11871      void     setbuf(FILE *restrict, char *restrict);
11872      int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11873      int      snprintf(char *restrict, size_t, const char *restrict, ...);
11874      int      sprintf(char *restrict, const char *restrict, ...);
11875      int      sscanf(const char *restrict, const char *restrict, ...);
11876 OB XSI    char      *tempnam(const char *, const char *);
11877      FILE     *tmpfile(void);
11878 OB      char      *tmpnam(char *);
11879      int      ungetc(int, FILE *);
11880 CX      int      vdprintf(int, const char *restrict, va_list);
11881      int      vfprintf(FILE *restrict, const char *restrict, va_list);
11882      int      vfprintf(FILE *restrict, const char *restrict, va_list);
11883      int      vprintf(const char *restrict, va_list);
11884      int      vscanf(const char *restrict, va_list);
11885      int      vsnprintf(char *restrict, size_t, const char *restrict,
11886                  va_list);
11887      int      vsprintf(char *restrict, const char *restrict, va_list);
11888      int      vsscanf(const char *restrict, const char *restrict, va_list);
11889 CX      Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

```

## 11890 APPLICATION USAGE

11891 Since standard I/O streams may use an underlying file descriptor to access the file associated  
11892 with a stream, application developers need to be aware that {FOPEN\_MAX} streams may not be  
11893 available if file descriptors are being used to access files that are not associated with streams.

## 11894 RATIONALE

11895 There is a conflict between the ISO C standard and the POSIX definition of the {TMP\_MAX}  
11896 macro that is addressed by ISO/IEC 9899:1999 standard, Defect Report 336. The POSIX standard  
11897 is in alignment with the public record of the response to the Defect Report. This change has not  
11898 yet been published as part of the ISO C standard.

11899 **FUTURE DIRECTIONS**

11900 None.

11901 **SEE ALSO**

11902 &lt;stdarg.h&gt;, &lt;stddef.h&gt;, &lt;sys/types.h&gt;

11903 XSH Section 2.2 (on page 468), *clearerr()*, *ctermid()*, *fclose()*, *fdopen()*, *feof()*, *ferror()*, *fflush()*,  
 11904 *fgetc()*, *fgetpos()*, *fgets()*, *fileno()*, *flockfile()*, *fmemopen()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*,  
 11905 *freopen()*, *fscanf()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getchar()*, *getc\_unlocked()*, *getdelim()*,  
 11906 *getopt()*, *gets()*, *open\_memstream()*, *pclose()*, *perror()*, *popen()*, *putc()*, *putchar()*, *puts()*, *remove()*,  
 11907 *rename()*, *rewind()*, *setbuf()*, *setvbuf()*, *stdin*, *system()*, *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*,  
 11908 *vfprintf()*, *vfscanf()*

11909 **CHANGE HISTORY**

11910 First released in Issue 1. Derived from Issue 1 of the SVID.

11911 **Issue 5**

11912 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11913 Large File System extensions are added.

11914 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as  
 11915 extensions and LEGACY.11916 The *cuserid()* and *getopt()* functions are marked LEGACY.11917 **Issue 6**11918 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed  
 11919 as they were previously marked LEGACY.11920 The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked  
 11921 LEGACY.

11922 Several functions are marked as part of the Thread-Safe Functions option.

11923 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the  
 11924 description of the *fpos\_t* type is now explicitly updated to exclude array types.

11925 Extensions beyond the ISO C standard are marked.

11926 **Issue 7**11927 Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the  
 11928 definition of *{TMP\_MAX}* with the ISO C standard.

11929 SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE.

11930 The *dprintf()*, *fmemopen()*, *getdelim()*, *getline()*, *open\_memstream()*, and *vdprintf()* functions are  
 11931 added from The Open Group Technical Standard, 2006, Extended API Set Part 1.11932 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 11933 Set Part 2.11934 The *gets()*, *tmpnam()*, and *tempnam()* functions and the *L\_tmpnam* macro are marked  
 11935 obsolescent.11936 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
 11937 for the *off\_t* type is added.

## Headers

## &lt;stdlib.h&gt;

## 11938 NAME

11939        **stdlib.h** — standard library definitions

## 11940 SYNOPSIS

11941        #include &lt;stdlib.h&gt;

## 11942 DESCRIPTION

11943 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 11944       Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 11945       enable the visibility of these symbols in this header.

11946       The <stdlib.h> header shall define the following macros which shall expand to integer constant  
 11947       expressions:

11948       EXIT\_FAILURE   Unsuccessful termination for *exit()*; evaluates to a non-zero value.

11949       EXIT\_SUCCESS   Successful termination for *exit()*; evaluates to 0.

11950       {RAND\_MAX}     Maximum value returned by *rand()*; at least 32 767.

11951       The <stdlib.h> header shall define the following macro which shall expand to a positive integer  
 11952       expression with type **size\_t**:

11953       {MB\_CUR\_MAX}   Maximum number of bytes in a character specified by the current locale  
 11954       (category *LC\_CTYPE*).

11955       The <stdlib.h> header shall define NULL as described in <stddef.h>.

11956       The <stdlib.h> header shall define the following data types through **typedef**:

11957       **div\_t**           Structure type returned by the *div()* function.

11958       **ldiv\_t**          Structure type returned by the *ldiv()* function.

11959       **lldiv\_t**         Structure type returned by the *lldiv()* function.

11960       **size\_t**          As described in <stddef.h>.

11961       **wchar\_t**         As described in <stddef.h>.

11962 CX       In addition, the <stdlib.h> header shall define the following symbolic constants and macros as  
 11963       described in <sys/wait.h>:

11964       WEXITSTATUS

11965       WIFEXITED

11966       WIFSIGNALED

11967       WIFSTOPPED

11968       WNOHANG

11969       WSTOPSIG

11970       WTERMSIG

11971       WUNTRACED

11972       The following shall be declared as functions and may also be defined as macros. Function  
 11973       prototypes shall be provided.

11974       void            \_Exit(int);

11975 XSI       long        a64l(const char \*);

11976       void            abort(void);

11977       int             abs(int);

11978       int             atexit(void (\*) (void));

11979       double          atof(const char \*);

```

11980     int          atoi(const char *);
11981     long         atol(const char *);
11982     long long    atoll(const char *);
11983     void         *bsearch(const void *, const void *, size_t, size_t,
11984                          int (*)(const void *, const void *));
11985     void         *calloc(size_t, size_t);
11986     div_t        div(int, int);
11987 XSI     double      drand48(void);
11988     double      erand48(unsigned short [3]);
11989     void         exit(int);
11990     void         free(void *);
11991     char        *getenv(const char *);
11992     int         getsubopt(char **, char *const *, char **);
11993 XSI     int         grantpt(int);
11994     char        *initstate(unsigned, char *, size_t);
11995     long        jrand48(unsigned short [3]);
11996     char        *l64a(long);
11997     long        labs(long);
11998 XSI     void        lcong48(unsigned short [7]);
11999     ldiv_t      ldiv(long, long);
12000     long long   llabs(long long);
12001     lldiv_t     lldiv(long long, long long);
12002 XSI     long        lrand48(void);
12003     void        *malloc(size_t);
12004     int         mblen(const char *, size_t);
12005     size_t      mbstowcs(wchar_t *restrict, const char *restrict, size_t);
12006     int         mbtowlc(wchar_t *restrict, const char *restrict, size_t);
12007 CX      char        *mkdtemp(char *);
12008     int         mkstemp(char *);
12009 XSI     long        mrand48(void);
12010     long        nrand48(unsigned short [3]);
12011 ADV     int         posix_memalign(void **, size_t, size_t);
12012 XSI     int         posix_openpt(int);
12013     char        *ptname(int);
12014     int         putenv(char *);
12015     void        qsort(void *, size_t, size_t, int (*)(const void *,
12016               const void *));
12017     int         rand(void);
12018 OB CX    int         rand_r(unsigned *);
12019 XSI     long        random(void);
12020     void        *realloc(void *, size_t);
12021 XSI     char        *realpath(const char *restrict, char *restrict);
12022     unsigned short *seed48(unsigned short [3]);
12023 CX      int         setenv(const char *, const char *, int);
12024 XSI     void        setkey(const char *);
12025     char        *setstate(char *);
12026     void        srand(unsigned);
12027 XSI     void        srand48(long);
12028     void        srandom(unsigned);
12029     double      strtod(const char *restrict, char **restrict);
12030     float       strtod(const char *restrict, char **restrict);
12031     long        strtol(const char *restrict, char **restrict, int);

```

## Headers

## &lt;stdlib.h&gt;

```

12032     long double   strtold(const char *restrict, char **restrict);
12033     long long     strtoll(const char *restrict, char **restrict, int);
12034     unsigned long strtoul(const char *restrict, char **restrict, int);
12035     unsigned long long
12036                 strtoull(const char *restrict, char **restrict, int);
12037     int           system(const char *);
12038 XSI     int           unlockpt(int);
12039 CX     int           unsetenv(const char *);
12040     size_t       wcstombs(char *restrict, const wchar_t *restrict, size_t);
12041     int          wctomb(char *, wchar_t);

```

12042 CX Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>, <limits.h>, <math.h>, and <sys/wait.h>.

## 12044 APPLICATION USAGE

12045 None.

## 12046 RATIONALE

12047 None.

## 12048 FUTURE DIRECTIONS

12049 None.

## 12050 SEE ALSO

12051 <limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>

12052 XSH Section 2.2 (on page 468), *\_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*,  
12053 *bsearch()*, *calloc()*, *div()*, *drand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *labs()*,  
12054 *ldiv()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkdtemp()*, *posix\_memalign()*, *posix\_openpt()*,  
12055 *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setenv()*, *setkey()*, *strtod()*, *strtol()*,  
12056 *strtoul()*, *system()*, *unlockpt()*, *unsetenv()*, *wcstombs()*, *wctomb()*

## 12057 CHANGE HISTORY

12058 First released in Issue 3.

## 12059 Issue 5

12060 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12061 The *ttyslot()* and *valloc()* functions are marked LEGACY.

12062 The type of the third argument to *initstate()* is changed from **int** to **size\_t**. The type of the return  
12063 value from *setstate()* is changed from **char** to **char \***, and the type of the first argument is  
12064 changed from **char \*** to **const char \***.

## 12065 Issue 6

12066 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be  
12067 consistent with the reference page.

12068 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be  
12069 consistent with the reference page.

12070 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

12071 Function prototypes for *setenv()* and *unsetenv()* are added.

12072 The *posix\_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

12073 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12074 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.

- 12075 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.
- 12076 Extensions beyond the ISO C standard are marked.
- 12077 **Issue 7**
- 12078 SD5-XBD-ERN-79 and SD5-XBD-ERN-105 are applied.
- 12079 The LEGACY functions are removed.
- 12080 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
- 12081
- 12082 The *rand\_r()* function is marked obsolescent.
- 12083 This reference page is clarified with respect to macros and symbolic constants.
- 12084 The type of the first argument to *setstate()* is changed from **const char \*** to **char \***.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;string.h&gt;

## 12085 NAME

12086 string.h — string operations

## 12087 SYNOPSIS

12088 #include &lt;string.h&gt;

## 12089 DESCRIPTION

12090 CX Some of the functionality described on this reference page extends the ISO C standard.  
 12091 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 12092 enable the visibility of these symbols in this header.

12093 The &lt;string.h&gt; header shall define NULL and size\_t as described in &lt;stddef.h&gt;.

12094 CX The &lt;string.h&gt; header shall define the locale\_t type as described in &lt;locale.h&gt;.

12095 The following shall be declared as functions and may also be defined as macros. Function  
 12096 prototypes shall be provided for use with ISO C standard compilers.

12097 XSI void \*memcpy(void \*restrict, const void \*restrict, int, size\_t);  
 12098 void \*memchr(const void \*, int, size\_t);  
 12099 int memcmp(const void \*, const void \*, size\_t);  
 12100 void \*memcpy(void \*restrict, const void \*restrict, size\_t);  
 12101 void \*memmove(void \*, const void \*, size\_t);  
 12102 void \*memset(void \*, int, size\_t);  
 12103 CX char \*strcpy(char \*restrict, const char \*restrict);  
 12104 char \*strncpy(char \*restrict, const char \*restrict, size\_t);  
 12105 char \*strcat(char \*restrict, const char \*restrict);  
 12106 char \*strchr(const char \*, int);  
 12107 int strcmp(const char \*, const char \*);  
 12108 int strcoll(const char \*, const char \*);  
 12109 CX int strcoll\_l(const char \*, const char \*, locale\_t);  
 12110 char \*strcpy(char \*restrict, const char \*restrict);  
 12111 size\_t strcspn(const char \*, const char \*);  
 12112 CX char \*strdup(const char \*);  
 12113 char \*strerror(int);  
 12114 CX char \*strerror\_l(int, locale\_t);  
 12115 int strerror\_r(int, char \*, size\_t);  
 12116 size\_t strlen(const char \*);  
 12117 char \*strncat(char \*restrict, const char \*restrict, size\_t);  
 12118 int strncmp(const char \*, const char \*, size\_t);  
 12119 char \*strncpy(char \*restrict, const char \*restrict, size\_t);  
 12120 CX char \*strndup(const char \*, size\_t);  
 12121 size\_t strnlen(const char \*, size\_t);  
 12122 char \*strpbrk(const char \*, const char \*);  
 12123 char \*strrchr(const char \*, int);  
 12124 CX char \*strsignal(int);  
 12125 size\_t strspn(const char \*, const char \*);  
 12126 char \*strstr(const char \*, const char \*);  
 12127 char \*strtok(char \*restrict, const char \*restrict);  
 12128 CX char \*strtok\_r(char \*restrict, const char \*restrict, char \*\*restrict);  
 12129 size\_t strxfrm(char \*restrict, const char \*restrict, size\_t);  
 12130 CX size\_t strxfrm\_l(char \*restrict, const char \*restrict,  
 12131 size\_t, locale\_t);

**<string.h>**

12132 CX Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

12133 **APPLICATION USAGE**

12134 None.

12135 **RATIONALE**

12136 None.

12137 **FUTURE DIRECTIONS**

12138 None.

12139 **SEE ALSO**

12140 **<locale.h>**, **<stddef.h>**, **<sys/types.h>**

12141 XSH Section 2.2 (on page 468), *memcpy()*, *memchr()*, *memcmp()*, *memcpy()*, *memmove()*,  
 12142 *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strcspn()*, *strdup()*, *strerror()*, *strlen()*,  
 12143 *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strsignal()*, *strspn()*, *strstr()*, *strtok()*, *strxfrm()*

12144 **CHANGE HISTORY**

12145 First released in Issue 1. Derived from Issue 1 of the SVID.

12146 **Issue 5**

12147 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12148 **Issue 6**

12149 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

12150 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12151 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

12152 **Issue 7**

12153 SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror\_r()* function.

12154 The *stpcpy()*, *stpncpy()*, *strndup()*, *strlen()*, and *strsignal()* functions are added from The Open  
 12155 Group Technical Standard, 2006, Extended API Set Part 1.

12156 The *strcoll\_l()*, *strerror\_l()*, and *strxfrm\_l()* functions are added from The Open Group Technical  
 12157 Standard, 2006, Extended API Set Part 4.

12158 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
 12159 for the **locale\_t** type is added.

12160 **NAME**

12161 strings.h — string operations

12162 **SYNOPSIS**

12163 #include &lt;strings.h&gt;

12164 **DESCRIPTION**12165 The following shall be declared as functions and may also be defined as macros. Function  
12166 prototypes shall be provided for use with ISO C standard compilers.

```

12167 XSI int ffs(int);
12168 int strcasecmp(const char *, const char *);
12169 int strcasecmp_l(const char *, const char *, locale_t);
12170 int strncasecmp(const char *, const char *, size_t);
12171 int strncasecmp_l(const char *, const char *, size_t, locale_t);

```

12172 The <strings.h> header shall define the **locale\_t** type as described in <locale.h>.12173 The <strings.h> header shall define the **size\_t** type as described in <sys/types.h>.12174 **APPLICATION USAGE**

12175 None.

12176 **RATIONALE**

12177 None.

12178 **FUTURE DIRECTIONS**

12179 None.

12180 **SEE ALSO**

12181 &lt;locale.h&gt;, &lt;sys/types.h&gt;

12182 XSH *ffs()*, *strcasecmp()*12183 **CHANGE HISTORY**

12184 First released in Issue 4, Version 2.

12185 **Issue 6**12186 The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index()* to be  
12187 consistent with the reference page.12188 The *bcmp()*, *bcopy()*, *bzero()*, *index()*, and *rindex()* functions are marked LEGACY.12189 **Issue 7**12190 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

12191 The LEGACY functions are removed.

12192 The &lt;strings.h&gt; header is moved from the XSI option to the Base.

12193 The *strcasecmp\_l()* and *strncasecmp\_l()* functions are added from The Open Group Technical  
12194 Standard, 2006, Extended API Set Part 4.12195 A declaration for the **locale\_t** type is added.

**<stropts.h>**12196 **NAME**12197           stropts.h — STREAMS interface (**STREAMS**)12198 **SYNOPSIS**12199 OB XSR `#include <stropts.h>`12200 **DESCRIPTION**12201           The **<stropts.h>** header shall define the **bandinfo** structure, which shall include at least the  
12202 following members:12203           int                bi\_flag     Flushing type.  
12204           unsigned char    bi\_pri     Priority band.12205           The **<stropts.h>** header shall define the **strpeek** structure, which shall include at least the  
12206 following members:12207           struct strbuf    ctlbuf     The control portion of the message.  
12208           struct strbuf    databuf    The data portion of the message.  
12209           t\_uscalar\_t     flags     RS\_HIPRI or 0.12210           The **<stropts.h>** header shall define the **strbuf** structure, which shall include at least the  
12211 following members:12212           char   \*buf        Pointer to buffer.  
12213           int    len         Length of data.  
12214           int    maxlen     Maximum buffer length.12215           The **<stropts.h>** header shall define the **strfdinsert** structure, which shall include at least the  
12216 following members:12217           struct strbuf    ctlbuf     The control portion of the message.  
12218           struct strbuf    databuf    The data portion of the message.  
12219           int              fildes     File descriptor of the other STREAM.  
12220           t\_uscalar\_t     flags     RS\_HIPRI or 0.  
12221           int              offset     Relative location of the stored value.12222           The **<stropts.h>** header shall define the **striocctl** structure, which shall include at least the  
12223 following members:12224           int    ic\_cmd     *ioctl()* command.  
12225           char   \*ic\_dp     Pointer to buffer.  
12226           int    ic\_len     Length of data.  
12227           int    ic\_timeout   Timeout for response.12228           The **<stropts.h>** header shall define the **strrecvfd** structure, which shall include at least the  
12229 following members:12230           int    fd         Received file descriptor.  
12231           gid\_t  gid        GID of sender.  
12232           uid\_t  uid        UID of sender.12233           The **<stropts.h>** header shall define the **uid\_t** and **gid\_t** types through **typedef**, as described in  
12234 **<sys/types.h>**.12235           The **<stropts.h>** header shall define the **t\_scalar\_t** and **t\_uscalar\_t** types, respectively, as signed  
12236 and unsigned opaque types of equal length of at least 32 bits.12237           The **<stropts.h>** header shall define the **str\_list** structure, which shall include at least the  
12238 following members:

## Headers

## &lt;stropts.h&gt;

12239	struct str_mlist	*sl_modlist	STREAMS module names.
12240	int	sl_nmods	Number of STREAMS module names.
12241	The <stropts.h> header shall define the <b>str_mlist</b> structure, which shall include at least the		
12242	following member:		
12243	char	l_name[FMNAMESZ+1]	A STREAMS module name.
12244	The <stropts.h> header shall define at least the following symbolic constants for use as the		
12245	<i>request</i> argument to <i>ioctl()</i> :		
12246	I_ATMARK		Is the top message “marked”?
12247	I_CANPUT		Is a band writable?
12248	I_CKBAND		See if any messages exist in a band.
12249	I_FDINSERT		Send implementation-defined information about another STREAM.
12250	I_FIND		Look for a STREAMS module.
12251	I_FLUSH		Flush a STREAM.
12252	I_FLUSHBAND		Flush one band of a STREAM.
12253	I_GETBAND		Get the band of the top message on a STREAM.
12254	I_GETCLTIME		Get close time delay.
12255	I_GETSIG		Retrieve current notification signals.
12256	I_GRDOPT		Get the read mode.
12257	I_GWROPT		Get the write mode.
12258	I_LINK		Connect two STREAMs.
12259	I_LIST		Get all the module names on a STREAM.
12260	I_LOOK		Get the top module name.
12261	I_NREAD		Size the top message.
12262	I_PEEK		Peek at the top message on a STREAM.
12263	I_PLINK		Persistently connect two STREAMs.
12264	I_POP		Pop a STREAMS module.
12265	I_PUNLINK		Dismantle a persistent STREAMS link.
12266	I_PUSH		Push a STREAMS module.
12267	I_RECVFD		Get a file descriptor sent via I_SENDFD.
12268	I_SENDFD		Pass a file descriptor through a STREAMS pipe.
12269	I_SETCLTIME		Set close time delay.
12270	I_SETSIG		Ask for notification signals.
12271	I_SRDOPT		Set the read mode.
12272	I_STR		Send a STREAMS <i>ioctl()</i> .

## &lt;stropts.h&gt;

## Headers

12273	I_SWROPT	Set the write mode.
12274	I_UNLINK	Disconnect two STREAMs.
12275	The <stropts.h>	header shall define at least the following symbolic constant for use with
12276	I_LOOK:	
12277	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
12278	The <stropts.h>	header shall define at least the following symbolic constants for use with
12279	I_FLUSH:	
12280	FLUSHR	Flush read queues.
12281	FLUSHRW	Flush read and write queues.
12282	FLUSHW	Flush write queues.
12283	The <stropts.h>	header shall define at least the following symbolic constants for use with
12284	I_SETSIG:	
12285	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of
12286		SIGPOLL when a priority message reaches the front of the STREAM head read
12287		queue.
12288	S_ERROR	Notification of an error condition reaches the STREAM head.
12289	S_HANGUP	Notification of a hangup reaches the STREAM head.
12290	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12291	S_INPUT	A message, other than a high-priority message, has arrived at the head of a
12292		STREAM head read queue.
12293	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the
12294		front of the STREAM head read queue.
12295	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM
12296		head is no longer full. This notifies the process that there is room on the queue
12297		for sending (or writing) normal data downstream.
12298	S_RDBAND	A message with a non-zero priority band has arrived at the head of a
12299		STREAM head read queue.
12300	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a
12301		STREAM head read queue.
12302	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is
12303		no longer full.
12304	S_WRNORM	Equivalent to S_OUTPUT.
12305	The <stropts.h>	header shall define at least the following symbolic constant for use with
12306	I_PEEK:	
12307	RS_HIPRI	Only look for high-priority messages.
12308	The <stropts.h>	header shall define at least the following symbolic constants for use with
12309	I_SRDOPT:	
12310	RMSGD	Message-discard mode.

## Headers

## &lt;stropts.h&gt;

12311	RMSGN	Message-non-discard mode.
12312	RNORM	Byte-STREAM mode, the default.
12313	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12314	RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12315		
12316	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12317		
12318	The <stropts.h>	header shall define at least the following symbolic constant for use with
12319	L_SWOPT:	
12320	SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0 bytes occurs.
12321	The <stropts.h>	header shall define at least the following symbolic constants for use with
12322	L_ATMARK:	
12323	ANYMARK	Check if the message is marked.
12324	LASTMARK	Check if the message is the last one marked on the queue.
12325	The <stropts.h>	header shall define at least the following symbolic constant for use with
12326	L_UNLINK:	
12327	MUXID_ALL	Unlink all STREAMs linked to the STREAM associated with <i>fildev</i> .
12328	The <stropts.h>	header shall define the following symbolic constants for <i>getmsg()</i> , <i>getpmsg()</i> ,
12329	<i>putmsg()</i> , and <i>putpmsg()</i> :	
12330	MORECTL	More control information is left in message.
12331	MOREDATA	More data is left in message.
12332	MSG_ANY	Receive any message.
12333	MSG_BAND	Receive message from specified band.
12334	MSG_HIPRI	Send/receive high-priority message.
12335	The <stropts.h>	header may make visible all of the symbols from <unistd.h>.
12336	The following shall be declared as functions and may also be defined as macros. Function	
12337	prototypes shall be provided.	
12338	int	<i>fattach</i> (int, const char *);
12339	int	<i>idetach</i> (const char *);
12340	int	<i>getmsg</i> (int, struct strbuf *restrict, struct strbuf *restrict,
12341	int	*restrict);
12342	int	<i>getpmsg</i> (int, struct strbuf *restrict, struct strbuf *restrict,
12343	int	*restrict, int *restrict);
12344	int	<i>ioctl</i> (int, int, ...);
12345	int	<i>isastream</i> (int);
12346	int	<i>putmsg</i> (int, const struct strbuf *, const struct strbuf *, int);
12347	int	<i>putpmsg</i> (int, const struct strbuf *, const struct strbuf *, int,
12348	int);	

**<stropts.h>**

Headers

- 12349 **APPLICATION USAGE**  
12350 None.
- 12351 **RATIONALE**  
12352 None.
- 12353 **FUTURE DIRECTIONS**  
12354 None.
- 12355 **SEE ALSO**  
12356 [<sys/types.h>](#), [<unistd.h>](#)
- 12357 XSH *close()*, *fattach()*, *fcntl()*, *fdetach()*, *getmsg()*, *ioctl()*, *isastream()*, *open()*, *pipe()*, *read()*, *poll()*,  
12358 *putmsg()*, *signal()*, *write()*
- 12359 **CHANGE HISTORY**  
12360 First released in Issue 4, Version 2.
- 12361 **Issue 5**  
12362 The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to  
12363 **t\_uscalar\_t**.
- 12364 **Issue 6**  
12365 This header is marked as part of the XSI STREAMS Option Group.  
12366 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.
- 12367 **Issue 7**  
12368 SD5-XBD-ERN-87 is applied, correcting an error in the **strrecvfd** structure.  
12369 The **<stropts.h>** header is marked obsolescent.  
12370 This reference page is clarified with respect to macros and symbolic constants.

12371 **NAME**

12372 sys/ipc.h — XSI interprocess communication access structure

12373 **SYNOPSIS**12374 XSI 

```
#include <sys/ipc.h>
```

12375 **DESCRIPTION**

12376 The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC):  
 12377 messages, semaphores, and shared memory. All use a common structure type, **ipc\_perm**, to pass  
 12378 information used in determining permission to perform an IPC operation.

12379 The <sys/ipc.h> header shall define the **ipc\_perm** structure, which shall include the following  
 12380 members:

12381	uid_t	uid	Owner's user ID.
12382	gid_t	gid	Owner's group ID.
12383	uid_t	cuid	Creator's user ID.
12384	gid_t	cgid	Creator's group ID.
12385	mode_t	mode	Read/write permission.

12386 The <sys/ipc.h> header shall define the **uid\_t**, **gid\_t**, **mode\_t**, and **key\_t** types as described in  
 12387 <sys/types.h>.

12388 The <sys/ipc.h> header shall define the following symbolic constants.

12389 Mode bits:

12390	IPC_CREAT	Create entry if key does not exist.
12391	IPC_EXCL	Fail if key exists.
12392	IPC_NOWAIT	Error if request must wait.

12393 Keys:

12394	IPC_PRIVATE	Private key.
-------	-------------	--------------

12395 Control commands:

12396	IPC_RMID	Remove identifier.
12397	IPC_SET	Set options.
12398	IPC_STAT	Get options.

12399 The following shall be declared as a function and may also be defined as a macro. A function  
 12400 prototype shall be provided.

```
12401 key_t ftok(const char *, int);
```

12402 **APPLICATION USAGE**

12403 None.

12404 **RATIONALE**

12405 None.

12406 **FUTURE DIRECTIONS**

12407 None.

**<sys/ipc.h>***Headers*12408 **SEE ALSO**12409 [<sys/types.h>](#)12410 [XSH.ftok\(\)](#)12411 **CHANGE HISTORY**

12412 First released in Issue 2. Derived from System V Release 2.0.

12413 **Issue 7**

12414 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;sys/mman.h&gt;

## 12415 NAME

12416 sys/mman.h — memory management declarations

## 12417 SYNOPSIS

12418 #include <sys/mman.h>

## 12419 DESCRIPTION

12420 The <sys/mman.h> header shall define the following symbolic constants for use as protection  
12421 options:

12422 PROT\_EXEC Page can be executed.

12423 PROT\_NONE Page cannot be accessed.

12424 PROT\_READ Page can be read.

12425 PROT\_WRITE Page can be written.

12426 The <sys/mman.h> header shall define the following symbolic constants for use as flag options:

12427 MAP\_FIXED Interpret *addr* exactly.

12428 MAP\_PRIVATE Changes are private.

12429 MAP\_SHARED Share changes.

12430 XSI|SIO The <sys/mman.h> header shall define the following symbolic constants for the *msync()*  
12431 function:

12432 MS\_ASYNC Perform asynchronous writes.

12433 MS\_INVALIDATE Invalidate mappings.

12434 MS\_SYNC Perform synchronous writes.

12435 ML The <sys/mman.h> header shall define the following symbolic constants for the *mlockall()*  
12436 function:

12437 MCL\_CURRENT Lock currently mapped pages.

12438 MCL\_FUTURE Lock pages that become mapped.

12439 The <sys/mman.h> header shall define the symbolic constant MAP\_FAILED which shall have  
12440 type **void \*** and shall be used to indicate a failure from the *mmap()* function .

12441 ADV If the Advisory Information option is supported, the <sys/mman.h> header shall define  
12442 symbolic constants for the *advice* argument to the *posix\_madvise()* function as follows:

12443 POSIX\_MADV\_DONTNEED

12444 The application expects that it will not access the specified range in the near future.

12445 POSIX\_MADV\_NORMAL

12446 The application has no advice to give on its behavior with respect to the specified range. It  
12447 is the default characteristic if no advice is given for a range of memory.

12448 POSIX\_MADV\_RANDOM

12449 The application expects to access the specified range in a random order.

12450 POSIX\_MADV\_SEQUENTIAL

12451 The application expects to access the specified range sequentially from lower addresses to  
12452 higher addresses.

## &lt;sys/mman.h&gt;

12453		POSIX_MADV_WILLNEED	
12454			The application expects to access the specified range in the near future.
12455	TYM	The <sys/mman.h> header shall define the following symbolic constants for use as flags for the <i>posix_typed_mem_open()</i> function:	
12456			
12457		POSIX_TYPED_MEM_ALLOCATE	
12458			Allocate on <i>mmap()</i> .
12459		POSIX_TYPED_MEM_ALLOCATE_CONTIG	
12460			Allocate contiguously on <i>mmap()</i> .
12461		POSIX_TYPED_MEM_MAP_ALLOCATABLE	
12462			Map on <i>mmap()</i> , without affecting allocatability.
12463		The <sys/mman.h> header shall define the <b>mode_t</b> , <b>off_t</b> , and <b>size_t</b> types as described in <sys/types.h>.	
12464			
12465	TYM	The <sys/mman.h> header shall define the <b>posix_typed_mem_info</b> structure, which shall include at least the following member:	
12466			
12467		size_t posix_tmi_length	Maximum length which may be allocated from a typed memory object.
12468			
12469		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
12470			
12471	MLR	int	mlock(const void *, size_t);
12472	ML	int	mlockall(int);
12473		void *	mmap(void *, size_t, int, int, int, off_t);
12474		int	mprotect(void *, size_t, int);
12475	XSI SIO	int	msync(void *, size_t, int);
12476	MLR	int	munlock(const void *, size_t);
12477	ML	int	munlockall(void);
12478		int	munmap(void *, size_t);
12479	ADV	int	posix_madvise(void *, size_t, int);
12480	TYM	int	posix_mem_offset(const void *restrict, size_t, off_t *restrict, size_t *restrict, int *restrict);
12481			
12482		int	posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
12483		int	posix_typed_mem_open(const char *, int, int);
12484	SHM	int	shm_open(const char *, int, mode_t);
12485		int	shm_unlink(const char *);

12486 **APPLICATION USAGE**

12487 None.

12488 **RATIONALE**

12489 None.

12490 **FUTURE DIRECTIONS**

12491 None.

12492 **SEE ALSO**

12493 &lt;sys/types.h&gt;

12494 XSH *mlock()*, *mlockall()*, *mmap()*, *mprotect()*, *msync()*, *munmap()*, *posix\_madvise()*,  
 12495 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *posix\_typed\_mem\_open()*, *shm\_open()*,  
 12496 *shm\_unlink()*

12497 **CHANGE HISTORY**

12498 First released in Issue 4, Version 2.

12499 **Issue 5**

12500 Updated for alignment with the POSIX Realtime Extension.

12501 **Issue 6**

12502 The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped  
 12503 Files, Process Memory Locking, or Shared Memory Objects options.

12504 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 12505 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header  
 12506 line, as well as for other lines.
- 12507 • The POSIX\_TYPED\_MEM\_ALLOCATE, POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG,  
 12508 and POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flags are added.
- 12509 • The **posix\_tmi\_length** structure is added.
- 12510 • The *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, and *posix\_typed\_mem\_open()* functions  
 12511 are added.

12512 The **restrict** keyword is added to the prototype for *posix\_mem\_offset()*.12513 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix\_madvise()*.

12514 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and  
 12515 shading errors for the *mlock()* and *munlock()* functions.

12516 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code  
 12517 for the *mmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

12518 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code  
 12519 for the *munmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

12520 **Issue 7**

12521 SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.

12522 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to  
 12523 the Base.

12524 This reference page is clarified with respect to macros and symbolic constants.

**<sys/msg.h>**12525 **NAME**

12526 sys/msg.h — XSI message queue structures

12527 **SYNOPSIS**12528 XSI `#include <sys/msg.h>`12529 **DESCRIPTION**12530 The **<sys/msg.h>** header shall define the following data types through **typedef**:12531 **msgqnum\_t** Used for the number of messages in the message queue.12532 **msglen\_t** Used for the number of bytes allowed in a message queue.12533 These types shall be unsigned integer types that are able to store values at least as large as a type  
12534 **unsigned short**.12535 The **<sys/msg.h>** header shall define the following symbolic constant as a message operation  
12536 flag:12537 **MSG\_NOERROR** No error if big message.12538 The **<sys/msg.h>** header shall define the **msqid\_ds** structure, which shall include the following  
12539 members:

12540	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12541	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12542	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12543	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <code>msgsnd()</code> .
12544	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <code>msgrcv()</code> .
12545	<code>time_t</code>	<code>msg_stime</code>	Time of last <code>msgsnd()</code> .
12546	<code>time_t</code>	<code>msg_rtime</code>	Time of last <code>msgrcv()</code> .
12547	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12548 The **<sys/msg.h>** header shall define the **pid\_t**, **size\_t**, **ssize\_t**, and **time\_t** types as described in  
12549 **<sys/types.h>**.12550 The following shall be declared as functions and may also be defined as macros. Function  
12551 prototypes shall be provided.

```

12552 int      msgctl(int, int, struct msqid_ds *);
12553 int      msgget(key_t, int);
12554 ssize_t  msgrcv(int, void *, size_t, long, int);
12555 int      msgsnd(int, const void *, size_t, int);

```

12556 In addition, the **<sys/msg.h>** header shall include the **<sys/ipc.h>** header.12557 **APPLICATION USAGE**

12558 None.

12559 **RATIONALE**

12560 None.

12561 **FUTURE DIRECTIONS**

12562 None.

12563 **SEE ALSO**12564 **<sys/ipc.h>**, **<sys/types.h>**12565 XSH `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

*Headers*

&lt;sys/msg.h&gt;

12566 **CHANGE HISTORY**

12567 First released in Issue 2. Derived from System V Release 2.0.

12568 **Issue 7**

12569 Austin Group Interpretation 1003.1-2001 #179 is applied.

12570 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## &lt;sys/resource.h&gt;

## 12571 NAME

12572 sys/resource.h — definitions for XSI resource operations

## 12573 SYNOPSIS

12574 XSI `#include <sys/resource.h>`

## 12575 DESCRIPTION

12576 The <sys/resource.h> header shall define the following symbolic constants as possible values of  
12577 the *which* argument of *getpriority()* and *setpriority()*:12578 PRIO\_PROCESS Identifies the *who* argument as a process ID.12579 PRIO\_PGRP Identifies the *who* argument as a process group ID.12580 PRIO\_USER Identifies the *who* argument as a user ID.12581 The <sys/resource.h> header shall define the following type through **typedef**:12582 **rlim\_t** Unsigned integer type used for limit values.12583 The <sys/resource.h> header shall define the following symbolic constants, which shall have  
12584 values suitable for use in **#if** preprocessing directives:12585 RLIM\_INFINITY A value of **rlim\_t** indicating no limit.12586 RLIM\_SAVED\_MAX A value of type **rlim\_t** indicating an unrepresentable saved hard  
12587 limit.12588 RLIM\_SAVED\_CUR A value of type **rlim\_t** indicating an unrepresentable saved soft limit.12589 On implementations where all resource limits are representable in an object of type **rlim\_t**,  
12590 RLIM\_SAVED\_MAX and RLIM\_SAVED\_CUR need not be distinct from RLIM\_INFINITY.12591 The <sys/resource.h> header shall define the following symbolic constants as possible values of  
12592 the *who* parameter of *getrusage()*:

12593 RUSAGE\_SELF Returns information about the current process.

12594 RUSAGE\_CHILDREN Returns information about children of the current process.

12595 The <sys/resource.h> header shall define the **rlimit** structure, which shall include at least the  
12596 following members:12597 `rlim_t rlim_cur` The current (soft) limit.12598 `rlim_t rlim_max` The hard limit.12599 The <sys/resource.h> header shall define the **rusage** structure, which shall include at least the  
12600 following members:12601 `struct timeval ru_utime` User time used.12602 `struct timeval ru_stime` System time used.12603 The <sys/resource.h> header shall define the **timeval** structure as described in <sys/time.h>.12604 The <sys/resource.h> header shall define the following symbolic constants as possible values for  
12605 the *resource* argument of *getrlimit()* and *setrlimit()*:12606 RLIMIT\_CORE Limit on size of **core** file.

12607 RLIMIT\_CPU Limit on CPU time per process.

## Headers

## &lt;sys/resource.h&gt;

12608	RLIMIT_DATA	Limit on data segment size.
12609	RLIMIT_FSIZE	Limit on file size.
12610	RLIMIT_NOFILE	Limit on number of open files.
12611	RLIMIT_STACK	Limit on stack size.
12612	RLIMIT_AS	Limit on address space size.
12613	The following shall be declared as functions and may also be defined as macros. Function	
12614	prototypes shall be provided.	
12615	<code>int getpriority(int, id_t);</code>	
12616	<code>int getrlimit(int, struct rlimit *);</code>	
12617	<code>int getrusage(int, struct rusage *);</code>	
12618	<code>int setpriority(int, id_t, int);</code>	
12619	<code>int setrlimit(int, const struct rlimit *);</code>	
12620	The <sys/resource.h> header shall define the <code>id_t</code> type through <code>typedef</code> , as described in	
12621	<sys/types.h>.	
12622	Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.	
12623	<b>APPLICATION USAGE</b>	
12624	None.	
12625	<b>RATIONALE</b>	
12626	None.	
12627	<b>FUTURE DIRECTIONS</b>	
12628	None.	
12629	<b>SEE ALSO</b>	
12630	<sys/time.h>, <sys/types.h>	
12631	XSH <code>getpriority()</code> , <code>getrlimit()</code> , <code>getrusage()</code>	
12632	<b>CHANGE HISTORY</b>	
12633	First released in Issue 4, Version 2.	
12634	<b>Issue 5</b>	
12635	Large File System extensions are added.	
12636	<b>Issue 7</b>	
12637	This reference page is clarified with respect to macros and symbolic constants.	

**<sys/select.h>**12638 **NAME**

12639 sys/select.h — select types

12640 **SYNOPSIS**

12641 #include &lt;sys/select.h&gt;

12642 **DESCRIPTION**12643 The **<sys/select.h>** header shall define the **timeval** structure, which shall include at least the  
12644 following members:12645 time\_t tv\_sec Seconds.  
12646 suseconds\_t tv\_usec Microseconds.12647 The **<sys/select.h>** header shall define the **time\_t** and **suseconds\_t** types as described in  
12648 **<sys/types.h>**.12649 The **<sys/select.h>** header shall define the **sigset\_t** type as described in **<signal.h>**.12650 The **<sys/select.h>** header shall define the **timespec** structure as described in **<time.h>**.12651 The **<sys/select.h>** header shall define the **fd\_set** type as a structure.12652 The **<sys/select.h>** header shall define the following symbolic constant, which shall have a value  
12653 suitable for use in **#if** preprocessing directives:12654 **FD\_SETSIZE** Maximum number of file descriptors in an **fd\_set** structure.12655 The following shall be declared as functions, defined as macros, or both. If functions are  
12656 declared, function prototypes shall be provided.12657 void FD\_CLR(int, fd\_set \*);  
12658 int FD\_ISSET(int, fd\_set \*);  
12659 void FD\_SET(int, fd\_set \*);  
12660 void FD\_ZERO(fd\_set \*);12661 If implemented as macros, these may evaluate their arguments more than once, so applications  
12662 should ensure that the arguments they supply are never expressions with side-effects.12663 The following shall be declared as functions and may also be defined as macros. Function  
12664 prototypes shall be provided.12665 int pselect(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
12666 const struct timespec \*restrict, const sigset\_t \*restrict);  
12667 int select(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
12668 struct timeval \*restrict);12669 Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers  
12670 **<signal.h>** and **<time.h>**.12671 **APPLICATION USAGE**

12672 None.

12673 **RATIONALE**

12674 None.

12675 **FUTURE DIRECTIONS**

12676 None.

12677 **SEE ALSO**12678 [<signal.h>](#), [<sys/time.h>](#), [<sys/types.h>](#), [<time.h>](#)12679 XSH *pselect()*12680 **CHANGE HISTORY**

12681 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12682 The requirement for the **fd\_set** structure to have a member *fds\_bits* has been removed as per The  
12683 Open Group Base Resolution bwg2001-005.12684 **Issue 7**

12685 SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

12686 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## &lt;sys/sem.h&gt;

## 12687 NAME

12688 sys/sem.h — XSI semaphore facility

## 12689 SYNOPSIS

12690 XSI `#include <sys/sem.h>`

## 12691 DESCRIPTION

12692 The <sys/sem.h> header shall define the following symbolic constant for use as a semaphore  
12693 operation flag:

12694 SEM\_UNDO Set up adjust on exit entry.

12695 The <sys/sem.h> header shall define the following symbolic constants for use as commands for  
12696 the *semctl()* function:12697 GETNCNT Get *semncnt*.12698 GETPID Get *sempid*.12699 GETVAL Get *semval*.12700 GETALL Get all cases of *semval*.12701 GETZCNT Get *semzcnt*.12702 SETVAL Set *semval*.12703 SETALL Set all cases of *semval*.12704 The <sys/sem.h> header shall define the **semid\_ds** structure, which shall include the following  
12705 members:

12706	<code>struct ipc_perm</code>	<code>sem_perm</code>	Operation permission structure.
12707	<code>unsigned short</code>	<code>sem_nsems</code>	Number of semaphores in set.
12708	<code>time_t</code>	<code>sem_otime</code>	Last <i>semop()</i> time.
12709	<code>time_t</code>	<code>sem_ctime</code>	Last time changed by <i>semctl()</i> .

12710 The <sys/sem.h> header shall define the **pid\_t**, **size\_t**, and **time\_t** types as described in  
12711 <sys/types.h>.12712 A semaphore shall be represented by an anonymous structure, which shall include the following  
12713 members:

12714	<code>unsigned short</code>	<code>semval</code>	Semaphore value.
12715	<code>pid_t</code>	<code>sempid</code>	Process ID of last operation.
12716	<code>unsigned short</code>	<code>semncnt</code>	Number of processes waiting for <i>semval</i> to become greater than current value.
12717			
12718	<code>unsigned short</code>	<code>semzcnt</code>	Number of processes waiting for <i>semval</i> to become 0.
12719			

12720 The <sys/sem.h> header shall define the **sembuf** structure, which shall include the following  
12721 members:

12722	<code>unsigned short</code>	<code>sem_num</code>	Semaphore number.
12723	<code>short</code>	<code>sem_op</code>	Semaphore operation.
12724	<code>short</code>	<code>sem_flg</code>	Operation flags.

12725 The following shall be declared as functions and may also be defined as macros. Function  
12726 prototypes shall be provided.12727 `int semctl(int, int, int, ...);`

*Headers***<sys/sem.h>**

12728           int     semget(key\_t, int, int);  
12729           int     semop(int, struct sembuf \*, size\_t);

12730           In addition, the **<sys/sem.h>** header shall include the **<sys/ipc.h>** header.

**12731 APPLICATION USAGE**

12732           None.

**12733 RATIONALE**

12734           None.

**12735 FUTURE DIRECTIONS**

12736           None.

**12737 SEE ALSO**

12738           [<sys/ipc.h>](#), [<sys/types.h>](#)

12739           XSH *semctl()*, *semget()*, *semop()*

**12740 CHANGE HISTORY**

12741           First released in Issue 2. Derived from System V Release 2.0.

**12742 Issue 7**

12743           Austin Group Interpretation 1003.1-2001 #179 is applied.

12744           This reference page is clarified with respect to macros and symbolic constants.

**<sys/shm.h>**12745 **NAME**

12746 sys/shm.h — XSI shared memory facility

12747 **SYNOPSIS**12748 XSI `#include <sys/shm.h>`12749 **DESCRIPTION**12750 The **<sys/shm.h>** header shall define the following symbolic constants:

12751 SHM\_RDONLY Attach read-only (else read-write).

12752 SHM\_RND Round attach address to SHMLBA.

12753 SHMLBA Segment low boundary address multiple.

12754 The **<sys/shm.h>** header shall define the following data types through **typedef**:12755 **shmatt\_t** Unsigned integer used for the number of current attaches that must be able to  
12756 store values at least as large as a type **unsigned short**.12757 The **<sys/shm.h>** header shall define the **shmid\_ds** structure, which shall include the following  
12758 members:

12759	<code>struct ipc_perm</code>	<code>shm_perm</code>	Operation permission structure.
12760	<code>size_t</code>	<code>shm_segsz</code>	Size of segment in bytes.
12761	<code>pid_t</code>	<code>shm_lpid</code>	Process ID of last shared memory operation.
12762	<code>pid_t</code>	<code>shm_cpid</code>	Process ID of creator.
12763	<code>shmatt_t</code>	<code>shm_nattch</code>	Number of current attaches.
12764	<code>time_t</code>	<code>shm_atime</code>	Time of last <code>shmat()</code> .
12765	<code>time_t</code>	<code>shm_dtime</code>	Time of last <code>shmdt()</code> .
12766	<code>time_t</code>	<code>shm_ctime</code>	Time of last change by <code>shmctl()</code> .

12767 The **<sys/shm.h>** header shall define the **pid\_t**, **size\_t**, and **time\_t** types as described in  
12768 **<sys/types.h>**.12769 The following shall be declared as functions and may also be defined as macros. Function  
12770 prototypes shall be provided.

```

12771 void *shmat(int, const void *, int);
12772 int shmctl(int, int, struct shmid_ds *);
12773 int shmdt(const void *);
12774 int shmget(key_t, size_t, int);

```

12775 In addition, the **<sys/shm.h>** header shall include the **<sys/ipc.h>** header.12776 **APPLICATION USAGE**

12777 None.

12778 **RATIONALE**

12779 None.

12780 **FUTURE DIRECTIONS**

12781 None.

12782 **SEE ALSO**12783 [<sys/ipc.h>](#), [<sys/types.h>](#)12784 XSH `shmat()`, `shmctl()`, `shmdt()`, `shmget()`

12785 **CHANGE HISTORY**

12786 First released in Issue 2. Derived from System V Release 2.0.

12787 **Issue 5**

12788 The type of *shm\_segsz* is changed from **int** to **size\_t**.

12789 **Issue 7**

12790 Austin Group Interpretation 1003.1-2001 #179 is applied.

12791 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## &lt;sys/socket.h&gt;

## 12792 NAME

12793 sys/socket.h — main sockets header

## 12794 SYNOPSIS

12795 #include &lt;sys/socket.h&gt;

## 12796 DESCRIPTION

12797 The <sys/socket.h> header shall define the **socklen\_t** type, which is an integer type of width of  
12798 at least 32 bits; see APPLICATION USAGE.12799 The <sys/socket.h> header shall define the **sa\_family\_t** unsigned integer type.12800 The <sys/socket.h> header shall define the **sockaddr** structure, which shall include at least the  
12801 following members:12802 sa\_family\_t sa\_family Address family.  
12803 char sa\_data[] Socket address (variable-length data).12804 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,  
12805 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.12806 The <sys/socket.h> header shall define the **sockaddr\_storage** structure, which shall be:

- 12807
- Large enough to accommodate all supported protocol-specific address structures
  - Aligned at an appropriate boundary so that pointers to it can be cast as pointers to  
12808 protocol-specific address structures and used to access the fields of those structures  
12809 without alignment problems
- 12810

12811 The **sockaddr\_storage** structure shall include at least the following members:

12812 sa\_family\_t ss\_family

12813 When a **sockaddr\_storage** structure is cast as a **sockaddr** structure, the *ss\_family* field of the  
12814 **sockaddr\_storage** structure shall map onto the *sa\_family* field of the **sockaddr** structure. When a  
12815 **sockaddr\_storage** structure is cast as a protocol-specific address structure, the *ss\_family* field  
12816 shall map onto a field of that structure that is of type **sa\_family\_t** and that identifies the  
12817 protocol's address family.12818 The <sys/socket.h> header shall define the **msghdr** structure, which shall include at least the  
12819 following members:12820 void \*msg\_name Optional address.  
12821 socklen\_t msg\_namelen Size of address.  
12822 struct iovec \*msg\_iov Scatter/gather array.  
12823 int msg\_iovlen Members in *msg\_iov*.  
12824 void \*msg\_control Ancillary data; see below.  
12825 socklen\_t msg\_controllen Ancillary data buffer *len*.  
12826 int msg\_flags Flags on received message.12827 The **msghdr** structure is used to minimize the number of directly supplied parameters to the  
12828 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the  
12829 *recvmsg()* function and *value* only for the *sendmsg()* function.12830 The <sys/socket.h> header shall define the **iovec** structure as described in <sys/uiio.h>.12831 The <sys/socket.h> header shall define the **cmsghdr** structure, which shall include at least the  
12832 following members:12833 socklen\_t cmsg\_len Data byte count, including the **cmsghdr**.  
12834 int cmsg\_level Originating protocol.

- 12835           int            msg\_type    Protocol-specific type.
- 12836           The **msg\_hdr** structure is used for storage of ancillary data object information.
- 12837           Ancillary data consists of a sequence of pairs, each consisting of a **msg\_hdr** structure followed  
12838           by a data array. The data array contains the ancillary data message, and the **msg\_hdr** structure  
12839           contains descriptive information that allows an application to correctly parse the data.
- 12840           The values for *msg\_level* shall be legal values for the *level* argument to the *getsockopt()* and  
12841           *setsockopt()* functions. The system documentation shall specify the *msg\_type* definitions for the  
12842           supported protocols.
- 12843           Ancillary data is also possible at the socket level. The <sys/socket.h> header shall define the  
12844           following symbolic constant for use as the *msg\_type* value when *msg\_level* is SOL\_SOCKET:
- 12845           SCM\_RIGHTS       Indicates that the data array contains the access rights to be sent or  
12846                               received.
- 12847           The <sys/socket.h> header shall define the following macros to gain access to the data arrays in  
12848           the ancillary data associated with a message header:
- 12849           MSG\_DATA(*msg*)  
12850           If the argument is a pointer to a **msg\_hdr** structure, this macro shall return an unsigned  
12851           character pointer to the data array associated with the **msg\_hdr** structure.
- 12852           MSG\_NXTHDR(*mhdr, msg*)  
12853           If the first argument is a pointer to a **msg\_hdr** structure and the second argument is a pointer  
12854           to a **msg\_hdr** structure in the ancillary data pointed to by the *msg\_control* field of that  
12855           **msg\_hdr** structure, this macro shall return a pointer to the next **msg\_hdr** structure, or a null  
12856           pointer if this structure is the last **msg\_hdr** in the ancillary data.
- 12857           MSG\_FIRSTHDR(*mhdr*)  
12858           If the argument is a pointer to a **msg\_hdr** structure, this macro shall return a pointer to the  
12859           first **msg\_hdr** structure in the ancillary data associated with this **msg\_hdr** structure, or a null  
12860           pointer if there is no ancillary data associated with the **msg\_hdr** structure.
- 12861           The <sys/socket.h> header shall define the **linger** structure, which shall include at least the  
12862           following members:
- 12863           int    l\_onoff    Indicates whether linger option is enabled.  
12864           int    l\_linger   Linger time, in seconds.
- 12865           The <sys/socket.h> header shall define the following symbolic constants with distinct values:
- 12866           SOCK\_DGRAM      Datagram socket.
- 12867    RS    SOCK\_RAW        Raw Protocol Interface.
- 12868           SOCK\_SEQPACKET   Sequenced-packet socket.
- 12869           SOCK\_STREAM     Byte-stream socket.
- 12870           The <sys/socket.h> header shall define the following symbolic constant for use as the *level*  
12871           argument of *setsockopt()* and *getsockopt()*.
- 12872           SOL\_SOCKET      Options to be accessed at socket level, not protocol level.
- 12873           The <sys/socket.h> header shall define the following symbolic constants with distinct values for  
12874           use as the *option\_name* argument in *getsockopt()* or *setsockopt()* calls (see XSH Section 2.10.16, on  
12875           page 522):

## &lt;sys/socket.h&gt;

12876	SO_ACCEPTCONN	Socket is accepting connections.
12877	SO_BROADCAST	Transmission of broadcast messages is supported.
12878	SO_DEBUG	Debugging information is being recorded.
12879	SO_DONTROUTE	Bypass normal routing.
12880	SO_ERROR	Socket error status.
12881	SO_KEEPALIVE	Connections are kept alive with periodic messages.
12882	SO_LINGER	Socket lingers on close.
12883	SO_OOBINLINE	Out-of-band data is transmitted in line.
12884	SO_RCVBUF	Receive buffer size.
12885	SO_RCVLOWAT	Receive "low water mark".
12886	SO_RCVTIMEO	Receive timeout.
12887	SO_REUSEADDR	Reuse of local addresses is supported.
12888	SO_SNDBUF	Send buffer size.
12889	SO_SNDLOWAT	Send "low water mark".
12890	SO_SNDTIMEO	Send timeout.
12891	SO_TYPE	Socket type.
12892	The <sys/socket.h> header shall define the following symbolic constant for use as the maximum	
12893	<i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
12894	SOMAXCONN	The maximum <i>backlog</i> queue length.
12895	The <sys/socket.h> header shall define the following symbolic constants with distinct values for	
12896	use as the valid values for the <i>msg_flags</i> field in the <i>msghdr</i> structure, or the <i>flags</i> parameter in	
12897	<i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
12898	MSG_CTRUNC	Control data truncated.
12899	MSG_DONTROUTE	Send without using routing tables.
12900	MSG_EOR	Terminates a record (if supported by the protocol).
12901	MSG_OOB	Out-of-band data.
12902	MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-oriented socket that is no longer connected.
12903		
12904	MSG_PEEK	Leave received data in queue.
12905	MSG_TRUNC	Normal data truncated.
12906	MSG_WAITALL	Attempt to fill the read buffer.
12907	The <sys/socket.h> header shall define the following symbolic constants with distinct values:	
12908	AF_INET	Internet domain sockets for use with IPv4 addresses.
12909	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.
12910	AF_UNIX	UNIX domain sockets.

12911 AF\_UNSPEC Unspecified.

12912 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

12913 SHUT\_RD Disables further receive operations.

12914 SHUT\_RDWR Disables further send and receive operations.

12915 SHUT\_WR Disables further send operations.

12916 The <sys/socket.h> header shall define the **size\_t** and **ssize\_t** types as described in

12917 <sys/types.h>.

12918 The following shall be declared as functions and may also be defined as macros. Function

12919 prototypes shall be provided.

```

12920 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
12921 int bind(int, const struct sockaddr *, socklen_t);
12922 int connect(int, const struct sockaddr *, socklen_t);
12923 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12924 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12925 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12926 int listen(int, int);
12927 ssize_t recv(int, void *, size_t, int);
12928 ssize_t recvfrom(int, void *restrict, size_t, int,
12929 struct sockaddr *restrict, socklen_t *restrict);
12930 ssize_t recvmsg(int, struct msghdr *, int);
12931 ssize_t send(int, const void *, size_t, int);
12932 ssize_t sendmsg(int, const struct msghdr *, int);
12933 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
12934 socklen_t);
12935 int setsockopt(int, int, int, const void *, socklen_t);
12936 int shutdown(int, int);
12937 int socketatmark(int);
12938 int socket(int, int, int);
12939 int socketpair(int, int, int, int [2]);

```

12940 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uio.h>.

#### 12941 APPLICATION USAGE

12942 To forestall portability problems, it is recommended that applications not use values larger than

12943  $2^{31} - 1$  for the **socklen\_t** type.

12944 The **sockaddr\_storage** structure solves the problem of declaring storage for automatic variables

12945 which is both large enough and aligned enough for storing the socket address data structure of

12946 any family. For example, code with a file descriptor and without the context of the address

12947 family can pass a pointer to a variable of this type, where a pointer to a socket address structure

12948 is expected in calls such as *getpeername()*, and determine the address family by accessing the

12949 received content after the call.

12950 The example below illustrates a data structure which aligns on a 64-bit boundary. An

12951 implementation-defined field *\_ss\_align* following *\_ss\_pad1* is used to force a 64-bit alignment

12952 which covers proper alignment good enough for needs of at least **sockaddr\_in6** (IPv6) and

12953 **sockaddr\_in** (IPv4) address data structures. The size of padding field *\_ss\_pad1* depends on the

12954 chosen alignment boundary. The size of padding field *\_ss\_pad2* depends on the value of overall

12955 size chosen for the total size of the structure. This size and alignment are represented in the

12956 above example by implementation-defined (not required) constants *\_SS\_MAXSIZE* (chosen

## &lt;sys/socket.h&gt;

12957 value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived  
12958 value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The  
12959 implementation-defined definitions and structure field names above start with an <underscore>  
12960 to denote implementation private name space. Portable code is not expected to access or  
12961 reference those fields or constants.

```

12962 /*
12963  * Desired design of maximum size and alignment.
12964  */
12965 #define _SS_MAXSIZE 128
12966     /* Implementation-defined maximum size. */
12967 #define _SS_ALIGNSIZE (sizeof(int64_t))
12968     /* Implementation-defined desired alignment. */
12969
12970 /*
12971  * Definitions used for sockaddr_storage structure paddings design.
12972  */
12973 #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12974 #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t) + \
12975     _SS_PAD1SIZE + _SS_ALIGNSIZE))
12976 struct sockaddr_storage {
12977     sa_family_t ss_family; /* Address family. */
12978     /*
12979     * Following fields are implementation-defined.
12980     */
12981     char _ss_pad1[_SS_PAD1SIZE];
12982     /* 6-byte pad; this is to make implementation-defined
12983     pad up to alignment field that follows explicit in
12984     the data structure. */
12985     int64_t _ss_align; /* Field to force desired structure
12986     storage alignment. */
12987     char _ss_pad2[_SS_PAD2SIZE];
12988     /* 112-byte pad to achieve desired size,
12989     _SS_MAXSIZE value minus size of ss_family
12990     __ss_pad1, __ss_align fields is 112. */
12991 };

```

12991 **RATIONALE**

12992 None.

12993 **FUTURE DIRECTIONS**

12994 None.

12995 **SEE ALSO**

12996 <sys/types.h>, <sys/uiio.h>

12997 XSH *accept()*, *bind()*, *connect()*, *getpeername()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*,  
12998 *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*, *socketmark()*, *socket()*,  
12999 *socketpair()*

13000 **CHANGE HISTORY**

13001 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13002 The **restrict** keyword is added to the prototypes for *accept()*, *getpeername()*, *getsockname()*,  
13003 *getsockopt()*, and *recvfrom()*.

13004 **Issue 7**

13005 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the `ssize_t` type.

13006 SD5-XBD-ERN-62 is applied.

13007 The `MSG_NOSIGNAL` symbolic constant is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

13009 This reference page is clarified with respect to macros and symbolic constants, and a declaration for the `size_t` type is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## &lt;sys/stat.h&gt;

## 13011 NAME

13012 sys/stat.h — data returned by the `stat()` function

## 13013 SYNOPSIS

13014 #include <sys/stat.h>

## 13015 DESCRIPTION

13016 The <sys/stat.h> header shall define the structure of the data returned by the `fstat()`, `lstat()`, and  
13017 `stat()` functions.

13018 The <sys/stat.h> header shall define the `stat` structure, which shall include at least the following  
13019 members:

13020	<code>dev_t st_dev</code>	Device ID of device containing file.
13021	<code>ino_t st_ino</code>	File serial number.
13022	<code>mode_t st_mode</code>	Mode of file (see below).
13023	<code>nlink_t st_nlink</code>	Number of hard links to the file.
13024	<code>uid_t st_uid</code>	User ID of file.
13025	<code>gid_t st_gid</code>	Group ID of file.
13026	XSI <code>dev_t st_rdev</code>	Device ID (if file is character or block special).
13027	<code>off_t st_size</code>	For regular files, the file size in bytes. For symbolic links, the length in bytes of the 13028 pathname contained in the symbolic link.
13029		For a shared memory object, the length in bytes.
13030	SHM	For a typed memory object, the length in bytes.
13031	TYM	For other file types, the use of this field is 13032 unspecified.
13033		
13034	<code>struct timespec st_atim</code>	Last data access timestamp.
13035	<code>struct timespec st_mtim</code>	Last data modification timestamp.
13036	<code>struct timespec st_ctim</code>	Last file status change timestamp.
13037	XSI <code>blksize_t st_blksize</code>	A file system-specific preferred I/O block size 13038 for this object. In some file system types, this 13039 may vary from file to file.
13040	<code>blkcnt_t st_blocks</code>	Number of blocks allocated for this object.

13041 The `st_ino` and `st_dev` fields taken together uniquely identify the file within the system.

13042 The <sys/stat.h> header shall define the `blkcnt_t`, `blksize_t`, `dev_t`, `ino_t`, `mode_t`, `nlink_t`,  
13043 `uid_t`, `gid_t`, `off_t`, and `time_t` types as described in <sys/types.h>.

13044 The <sys/stat.h> header shall define the `timespec` structure as described in <time.h>. Times  
13045 shall be given in seconds since the Epoch.

13046 Which structure members have meaningful values depends on the type of file. For further  
13047 information, see the descriptions of `fstat()`, `lstat()`, and `stat()` in the System Interfaces volume of  
13048 POSIX.1-2008.

13049 For compatibility with earlier versions of this standard, the `st_atime` macro shall be defined with  
13050 the value `st_atim.tv_sec`. Similarly, `st_ctime` and `st_mtime` shall be defined as macros with the  
13051 values `st_ctim.tv_sec` and `st_mtim.tv_sec`, respectively.

13052 The <sys/stat.h> header shall define the following symbolic constants for the file types encoded  
13053 in type **mode\_t**. The values shall be suitable for use in **#if** preprocessing directives:

13054	XSI	<b>S_IFMT</b>	Type of file.
13055		<b>S_IFBLK</b>	Block special.
13056		<b>S_IFCHR</b>	Character special.
13057		<b>S_IFIFO</b>	FIFO special.
13058		<b>S_IFREG</b>	Regular.
13059		<b>S_IFDIR</b>	Directory.
13060		<b>S_IFLNK</b>	Symbolic link.
13061		<b>S_IFSOCK</b>	Socket.

13062 The <sys/stat.h> header shall define the following symbolic constants for the file mode bits  
13063 encoded in type **mode\_t**, with the indicated numeric values. These macros shall expand to an  
13064 expression which has a type that allows them to be used, either singly or OR'ed together, as the  
13065 third argument to *open()* without the need for a **mode\_t** cast. The values shall be suitable for use  
13066 in **#if** preprocessing directives.

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
XSI S_ISVTX	01000	On directories, restricted deletion flag.

13083 The following macros shall be provided to test whether a file is of the specified type. The value  
13084 *m* supplied to the macros is the value of *st\_mode* from a **stat** structure. The macro shall evaluate  
13085 to a non-zero value if the test is true; 0 if the test is false.

13086	<b>S_ISBLK</b> ( <i>m</i> )	Test for a block special file.
13087	<b>S_ISCHR</b> ( <i>m</i> )	Test for a character special file.
13088	<b>S_ISDIR</b> ( <i>m</i> )	Test for a directory.
13089	<b>S_ISFIFO</b> ( <i>m</i> )	Test for a pipe or FIFO special file.
13090	<b>S_ISREG</b> ( <i>m</i> )	Test for a regular file.

13091		S_ISLNK( <i>m</i> )	Test for a symbolic link.
13092		S_ISSOCK( <i>m</i> )	Test for a socket.
13093		The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a <b>stat</b> structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the <b>stat</b> structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
13094			
13095			
13096			
13097			
13098			
13099		S_TYPEISMQ( <i>buf</i> )	Test for a message queue.
13100		S_TYPEISSEM( <i>buf</i> )	Test for a semaphore.
13101		S_TYPEISSHM( <i>buf</i> )	Test for a shared memory object.
13102	TYM	The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a <b>stat</b> structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the <b>stat</b> structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
13103			
13104			
13105			
13106			
13107		S_TYPEISTMO( <i>buf</i> )	Test macro for a typed memory object.
13108		The <sys/stat.h> header shall define the following symbolic constants as distinct integer values outside of the range [0,999 999 999], for use with the <i>futimens()</i> and <i>utimensat()</i> functions:	
13109			
13110		UTIME_NOW	
13111		UTIME_OMIT	
13112		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
13113			
13114		int chmod(const char *, mode_t);	
13115		int fchmod(int, mode_t);	
13116		int fchmodat(int, const char *, mode_t, int);	
13117		int fstat(int, struct stat *);	
13118		int fstatat(int, const char *restrict, struct stat *restrict, int);	
13119		int futimens(int, const struct timespec [2]);	
13120		int lstat(const char *restrict, struct stat *restrict);	
13121		int mkdir(const char *, mode_t);	
13122		int mkdirat(int, const char *, mode_t);	
13123		int mkfifo(const char *, mode_t);	
13124		int mkfifoat(int, const char *, mode_t);	
13125	XSI	int mknod(const char *, mode_t, dev_t);	
13126		int mknodat(int, const char *, mode_t, dev_t);	
13127		int stat(const char *restrict, struct stat *restrict);	
13128		mode_t umask(mode_t);	
13129		int utimensat(int, const char *, const struct timespec [2], int);	

13130 **APPLICATION USAGE**

13131 Use of the macros is recommended for determining the type of a file.

13132 **RATIONALE**

13133 A conforming C-language application must include <sys/stat.h> for functions that have  
 13134 arguments or return values of type **mode\_t**, so that symbolic values for that type can be used.  
 13135 An alternative would be to require that these constants are also defined by including  
 13136 <sys/types.h>.

13137 The S\_ISUID and S\_ISGID bits may be cleared on any write, not just on *open()*, as some  
 13138 historical implementations do.

13139 System calls that update the time entry fields in the **stat** structure must be documented by the  
 13140 implementors. POSIX-conforming systems should not update the time entry fields for functions  
 13141 listed in the System Interfaces volume of POSIX.1-2008 unless the standard requires that they do,  
 13142 except in the case of documented extensions to the standard.

13143 Upon assignment, file timestamps are immediately converted to the resolution of the file system  
 13144 by truncation (i.e., the recorded time can be older than the actual time). For example, if the file  
 13145 system resolution is 1 microsecond, then a conforming *stat()* must always return an  
 13146 *st\_mtim.tv\_nsec* that is a multiple of 1000. Some older implementations returned higher-  
 13147 resolution timestamps while the *inode* information was cached, and then spontaneously  
 13148 truncated the *tv\_nsec* fields when they were stored to and retrieved from disk, but this behavior  
 13149 does not conform.

13150 Note that *st\_dev* must be unique within a Local Area Network (LAN) in a “system” made up of  
 13151 multiple computers’ file systems connected by a LAN.

13152 Networked implementations of a POSIX-conforming system must guarantee that all files visible  
 13153 within the file tree (including parts of the tree that may be remotely mounted from other  
 13154 machines on the network) on each individual processor are uniquely identified by the  
 13155 combination of the *st\_ino* and *st\_dev* fields.

13156 The unit for the *st\_blocks* member of the **stat** structure is not defined within POSIX.1-2008. In  
 13157 some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation  
 13158 between values of the *st\_blocks* and *st\_blksize*, and the *f\_bsize* (from <sys/statvfs.h>) structure  
 13159 members.

13160 Traditionally, some implementations defined the multiplier for *st\_blocks* in <sys/param.h> as the  
 13161 symbol *DEV\_BSIZE*.

13162 Some earlier versions of this standard did not specify values for the file mode bit macros. The  
 13163 expectation was that some implementors might choose to use a different encoding for these bits  
 13164 than the traditional one, and that new applications would use symbolic file modes instead of  
 13165 numeric. This version of the standard specifies the traditional encoding, in recognition that  
 13166 nearly 20 years after the first publication of this standard numeric file modes are still in  
 13167 widespread use by application developers, and that all conforming implementations still use the  
 13168 traditional encoding.

13169 **FUTURE DIRECTIONS**

13170 No new S\_IFMT symbolic names for the file type values of **mode\_t** will be defined by  
 13171 POSIX.1-2008; if new file types are required, they will only be testable through *S\_ISxx()* or  
 13172 *S\_TYPEISxxx()* macros instead.

13173 **SEE ALSO**13174 [<sys/statvfs.h>](#), [<sys/types.h>](#), [<time.h>](#)13175 XSH [chmod\(\)](#), [fchmod\(\)](#), [fstat\(\)](#), [fstatat\(\)](#), [futimens\(\)](#), [mkdir\(\)](#), [mkfifo\(\)](#), [mknod\(\)](#), [umask\(\)](#)13176 **CHANGE HISTORY**

13177 First released in Issue 1. Derived from Issue 1 of the SVID.

13178 **Issue 5**

13179 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

13180 The type of *st\_blksize* is changed from **long** to **blksize\_t**; the type of *st\_blocks* is changed from  
13181 **long** to **blkcnt\_t**.13182 **Issue 6**13183 The S\_TYPEISMQ(), S\_TYPEISSEM(), and S\_TYPEISSHM() macros are unconditionally  
13184 mandated.13185 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize\_t**  
13186 and **blkcnt\_t** have been described.13187 The following new requirements on POSIX implementations derive from alignment with the  
13188 Single UNIX Specification:

- 13189
- The **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**, **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types are mandated.

13190 S\_IFSOCK and S\_ISSOCK are added for sockets.

13191 The description of **stat** structure members is changed to reflect contents when file type is a  
13192 symbolic link.

13193 The test macro S\_TYPEISTMO is added for alignment with IEEE Std 1003.1j-2000.

13194 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.13195 The *lstat()* function is made mandatory.13196 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the  
13197 *st\_blocks* member of the **stat** structure to the RATIONALE.13198 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the  
13199 DESCRIPTION that the **timespec** structure may be defined as described in the **<time.h>** header.13200 **Issue 7**13201 SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the  
13202 interfaces should be consulted in order to determine which structure members have meaningful  
13203 values.13204 The [fchmodat\(\)](#), [fstatat\(\)](#), [mkdirat\(\)](#), [mkfifoat\(\)](#), [mknodat\(\)](#), and [utimensat\(\)](#) functions are added  
13205 from The Open Group Technical Standard, 2006, Extended API Set Part 2.13206 The [futimens\(\)](#) function is added.

13207 This reference page is clarified with respect to macros and symbolic constants.

13208 Changes are made related to support for finegrained timestamps and the **UTIME\_NOW** and  
13209 **UTIME\_OMIT** symbolic constants are added.

13210 **NAME**

13211 sys/statvfs.h — VFS File System information structure

13212 **SYNOPSIS**

13213 #include &lt;sys/statvfs.h&gt;

13214 **DESCRIPTION**13215 The <sys/statvfs.h> header shall define the **statvfs** structure, which shall include at least the  
13216 following members:

13217	unsigned long	f_bsize	File system block size.
13218	unsigned long	f_frsize	Fundamental file system block size.
13219	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
13220	fsblkcnt_t	f_bfree	Total number of free blocks.
13221	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
13222			
13223	fsfilcnt_t	f_files	Total number of file serial numbers.
13224	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
13225	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
13226			
13227	unsigned long	f_fsid	File system ID.
13228	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
13229	unsigned long	f_namemax	Maximum filename length.

13230 The <sys/statvfs.h> header shall define the **fsblkcnt\_t** and **fsfilcnt\_t** types as described in  
13231 <sys/types.h>.13232 The <sys/statvfs.h> header shall define the following symbolic constants for the *f\_flag* member:

13233 ST\_RDONLY Read-only file system.

13234 ST\_NOSUID Does not support the semantics of the ST\_ISUID and ST\_ISGID file mode bits.

13235 The following shall be declared as functions and may also be defined as macros. Function  
13236 prototypes shall be provided.13237 int fstatvfs(int, struct statvfs \*);  
13238 int statvfs(const char \*restrict, struct statvfs \*restrict);13239 **APPLICATION USAGE**

13240 None.

13241 **RATIONALE**

13242 None.

13243 **FUTURE DIRECTIONS**

13244 None.

13245 **SEE ALSO**

13246 &lt;sys/types.h&gt;

13247 XSH *fstatvfs()*13248 **CHANGE HISTORY**

13249 First released in Issue 4, Version 2.

13250 **Issue 5**13251 The type of *f\_blocks*, *f\_bfree*, and *f\_bavail* is changed from **unsigned long** to **fsblkcnt\_t**; the type of  
13252 *f\_files*, *f\_ffree*, and *f\_favail* is changed from **unsigned long** to **fsfilcnt\_t**.

## &lt;sys/statvfs.h&gt;

Headers

13253 **Issue 6**

13254 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt\_t**  
13255 and **fsfilcnt\_t** have been described.

13256 The **restrict** keyword is added to the prototype for *statvfs()*.

13257 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of  
13258 ST\_NOSUID from “Does not support *setuid()/setgid()* semantics” to “Does not support the  
13259 semantics of the ST\_ISUID and ST\_ISGID file mode bits”.

13260 **Issue 7**

13261 The <sys/statvfs.h> header is moved from the XSI option to the Base.

13262 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

13263 **NAME**

13264 sys/time.h — time types

13265 **SYNOPSIS**13266 XSI 

```
#include <sys/time.h>
```

13267 **DESCRIPTION**13268 The <sys/time.h> header shall define the **timeval** structure, which shall include at least the  
13269 following members:13270 time\_t tv\_sec Seconds.  
13271 suseconds\_t tv\_usec Microseconds.13272 OB The <sys/time.h> header shall define the **itimerval** structure, which shall include at least the  
13273 following members:13274 struct timeval it\_interval Timer interval.  
13275 struct timeval it\_value Current value.13276 The <sys/time.h> header shall define the **time\_t** and **suseconds\_t** types as described in  
13277 <sys/types.h>.13278 The <sys/time.h> header shall define the **fd\_set** type as described in <sys/select.h>.13279 OB The <sys/time.h> header shall define the following symbolic constants for the *which* argument of  
13280 *getitimer()* and *setitimer()*:13281 ITIMER\_REAL Decrements in real time.  
13282 ITIMER\_VIRTUAL Decrements in process virtual time.  
13283 ITIMER\_PROF Decrements both in process virtual time and when the system is running  
13284 on behalf of the process.

13285 The &lt;sys/time.h&gt; header shall define the following as described in &lt;sys/select.h&gt;:

13286 FD\_CLR()  
13287 FD\_ISSET()  
13288 FD\_SET()  
13289 FD\_ZERO()  
13290 FD\_SETSIZE13291 The following shall be declared as functions and may also be defined as macros. Function  
13292 prototypes shall be provided.13293 OB 

```
int getitimer(int, struct itimerval *);  
13294 int gettimeofday(struct timeval *restrict, void *restrict);  
13295 int setitimer(int, const struct itimerval *restrict,  
13296 struct itimerval *restrict);  
13297 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,  
13298 struct timeval *restrict);  
13299 int utimes(const char *, const struct timeval [2]);
```

13300 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>  
13301 header.

## &lt;sys/time.h&gt;

Headers

13302 **APPLICATION USAGE**

13303 None.

13304 **RATIONALE**

13305 None.

13306 **FUTURE DIRECTIONS**

13307 None.

13308 **SEE ALSO**13309 [<sys/select.h>](#), [<sys/types.h>](#)13310 XSH *futimens()*, *getitimer()*, *gettimeofday()*, *pselect()*13311 **CHANGE HISTORY**

13312 First released in Issue 4, Version 2.

13313 **Issue 5**13314 The type of *tv\_usec* is changed from **long** to **suseconds\_t**.13315 **Issue 6**13316 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.13317 The note is added that inclusion of this header may also make symbols visible from  
13318 [<sys/select.h>](#).13319 The *utimes()* function is marked LEGACY.13320 **Issue 7**

13321 This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;sys/times.h&gt;

13322 **NAME**

13323 sys/times.h — file access and modification times structure

13324 **SYNOPSIS**

13325 #include <sys/times.h>

13326 **DESCRIPTION**

13327 The <sys/times.h> header shall define the **tms** structure, which is returned by *times()* and shall  
13328 include at least the following members:

13329 clock\_t tms\_utime User CPU time.  
13330 clock\_t tms\_stime System CPU time.  
13331 clock\_t tms\_cutime User CPU time of terminated child processes.  
13332 clock\_t tms\_cstime System CPU time of terminated child processes.

13333 The <sys/times.h> header shall define the **clock\_t** type as described in <sys/types.h>.

13334 The following shall be declared as a function and may also be defined as a macro. A function  
13335 prototype shall be provided.

13336 clock\_t times(struct tms \*);

13337 **APPLICATION USAGE**

13338 None.

13339 **RATIONALE**

13340 None.

13341 **FUTURE DIRECTIONS**

13342 None.

13343 **SEE ALSO**

13344 <sys/types.h>

13345 XSH *times()*

13346 **CHANGE HISTORY**

13347 First released in Issue 1. Derived from Issue 1 of the SVID.

## &lt;sys/types.h&gt;

## 13348 NAME

13349 sys/types.h — data types

## 13350 SYNOPSIS

13351 #include &lt;sys/types.h&gt;

## 13352 DESCRIPTION

13353 The &lt;sys/types.h&gt; header shall define at least the following types:

13354	<b>blkcnt_t</b>	Used for file block counts.
13355	<b>blksize_t</b>	Used for block sizes.
13356	<b>clock_t</b>	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13357		
13358	<b>clockid_t</b>	Used for clock ID type in the clock and timer functions.
13359	<b>dev_t</b>	Used for device IDs.
13360	XSI <b>fsblkcnt_t</b>	Used for file system block counts.
13361	XSI <b>fsfilcnt_t</b>	Used for file system file counts.
13362	<b>gid_t</b>	Used for group IDs.
13363	<b>id_t</b>	Used as a general identifier; can be used to contain at least a <b>pid_t</b> , <b>uid_t</b> , or <b>gid_t</b> .
13364		
13365	<b>ino_t</b>	Used for file serial numbers.
13366	XSI <b>key_t</b>	Used for XSI interprocess communication.
13367	<b>mode_t</b>	Used for some file attributes.
13368	<b>nlink_t</b>	Used for link counts.
13369	<b>off_t</b>	Used for file sizes.
13370	<b>pid_t</b>	Used for process IDs and process group IDs.
13371	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
13372	<b>pthread_barrier_t</b>	Used to identify a barrier.
13373	<b>pthread_barrierattr_t</b>	Used to define a barrier attributes object.
13374	<b>pthread_cond_t</b>	Used for condition variables.
13375	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
13376	<b>pthread_key_t</b>	Used for thread-specific data keys.
13377	<b>pthread_mutex_t</b>	Used for mutexes.
13378	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
13379	<b>pthread_once_t</b>	Used for dynamic package initialization.
13380	<b>pthread_rwlock_t</b>	Used for read-write locks.
13381	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
13382	<b>pthread_spinlock_t</b>	Used to identify a spin lock.

13383		<b>pthread_t</b>	Used to identify a thread.
13384		<b>size_t</b>	Used for sizes of objects.
13385		<b>ssize_t</b>	Used for a count of bytes or an error indication.
13386	XSI	<b>suseconds_t</b>	Used for time in microseconds.
13387		<b>time_t</b>	Used for time in seconds.
13388		<b>timer_t</b>	Used for timer ID returned by <i>timer_create()</i> .
13389	OB TRC		Also used to identify a trace stream attributes object.
13390	OB TRC	<b>trace_event_id_t</b>	Used to identify a trace event type.
13391	OB TEF	<b>trace_event_set_t</b>	Used to identify a trace event type set.
13392	OB TRC	<b>trace_id_t</b>	Used to identify a trace stream.
13393		<b>uid_t</b>	Used for user IDs.
13394			All of the types shall be defined as arithmetic types of an appropriate length, with the following
13395			exceptions:
13396		<b>pthread_attr_t</b>	
13397		<b>pthread_barrier_t</b>	
13398		<b>pthread_barrierattr_t</b>	
13399		<b>pthread_cond_t</b>	
13400		<b>pthread_condattr_t</b>	
13401		<b>pthread_key_t</b>	
13402		<b>pthread_mutex_t</b>	
13403		<b>pthread_mutexattr_t</b>	
13404		<b>pthread_once_t</b>	
13405		<b>pthread_rwlock_t</b>	
13406		<b>pthread_rwlockattr_t</b>	
13407		<b>pthread_spinlock_t</b>	
13408		<b>pthread_t</b>	
13409	OB TRC	<b>trace_attr_t</b>	
13410		<b>trace_event_id_t</b>	
13411	OB TEF	<b>trace_event_set_t</b>	
13412	OB TRC	<b>trace_id_t</b>	
13413			Additionally:
13414			• <b>mode_t</b> shall be an integer type.
13415			• <b>nlink_t</b> , <b>uid_t</b> , <b>gid_t</b> , and <b>id_t</b> shall be integer types.
13416			• <b>blkcnt_t</b> and <b>off_t</b> shall be signed integer types.
13417	XSI		• <b>fsblkcnt_t</b> , <b>fsfilcnt_t</b> , and <b>ino_t</b> shall be defined as unsigned integer types.
13418			• <b>size_t</b> shall be an unsigned integer type.
13419			• <b>blksize_t</b> , <b>pid_t</b> , and <b>ssize_t</b> shall be signed integer types.
13420			• <b>time_t</b> and <b>clock_t</b> shall be integer or real-floating types.
13421			The type <b>ssize_t</b> shall be capable of storing values at least in the range $[-1, \{SSIZE\_MAX\}]$ .

13422	XSI	The type <b>suseconds_t</b> shall be a signed integer type capable of storing values at least in the range [-1, 1 000 000].
13423		
13424		The implementation shall support one or more programming environments in which the widths of <b>blksize_t</b> , <b>pid_t</b> , <b>size_t</b> , <b>ssize_t</b> , and <b>suseconds_t</b> are no greater than the width of type <b>long</b> .
13425		
13426		The names of these programming environments can be obtained using the <i>confstr()</i> function or the <i>getconf</i> utility.
13427		
13428		There are no defined comparison or assignment operators for the following types:
13429		<b>pthread_attr_t</b>
13430		<b>pthread_barrier_t</b>
13431		<b>pthread_barrierattr_t</b>
13432		<b>pthread_cond_t</b>
13433		<b>pthread_condattr_t</b>
13434		<b>pthread_mutex_t</b>
13435		<b>pthread_mutexattr_t</b>
13436		<b>pthread_rwlock_t</b>
13437		<b>pthread_rwlockattr_t</b>
13438		<b>pthread_spinlock_t</b>
13439	OB TRC	<b>trace_attr_t</b>

## 13440 APPLICATION USAGE

13441 None.

## 13442 RATIONALE

13443 None.

## 13444 FUTURE DIRECTIONS

13445 None.

## 13446 SEE ALSO

13447 [<time.h>](#)13448 XSH *confstr()*13449 XCU *getconf*

## 13450 CHANGE HISTORY

13451 First released in Issue 1. Derived from Issue 1 of the SVID.

## 13452 Issue 5

13453 The **clockid\_t** and **timer\_t** types are defined for alignment with the POSIX Realtime Extension.13454 The types **blkcnt\_t**, **blksize\_t**, **fsblkcnt\_t**, **fsfilcnt\_t**, and **suseconds\_t** are added.

13455 Large File System extensions are added.

13456 Updated for alignment with the POSIX Threads Extension.

## 13457 Issue 6

13458 The **pthread\_barrier\_t**, **pthread\_barrierattr\_t**, and **pthread\_spinlock\_t** types are added for alignment with IEEE Std 1003.1j-2000.13460 The margin code is changed from XSI to THR for the **pthread\_rwlock\_t** and **pthread\_rwlockattr\_t** types as Read-Write Locks have been absorbed into the POSIX Threads option. The threads types are marked THR.

## Headers

## &lt;sys/types.h&gt;

- 13463 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread\_t** to the list  
13464 of types that are not required to be arithmetic types, thus allowing **pthread\_t** to be defined as a  
13465 structure.
- 13466 **Issue 7**
- 13467 Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key\_t** to be an arithmetic  
13468 type.
- 13469 The Trace option types are marked obsolescent.
- 13470 The **clock\_t** and **id\_t** types are moved from the XSI option to the Base.
- 13471 The **pthread\_barrier\_t** and **pthread\_barrierattr\_t** types are moved from the Barriers option to  
13472 the Base.
- 13473 The **pthread\_spinlock\_t** type is moved from the Spin Locks option to the Base.
- 13474 Functionality relating to the Timers and Threads options is moved to the Base.

**<sys/uio.h>**

Headers

13475 **NAME**

13476 sys/uio.h — definitions for vector I/O operations

13477 **SYNOPSIS**13478 XSI `#include <sys/uio.h>`13479 **DESCRIPTION**13480 The **<sys/uio.h>** header shall define the **iovec** structure, which shall include at least the  
13481 following members:13482 `void *iov_base` Base address of a memory region for input or output.  
13483 `size_t iov_len` The size of the memory pointed to by *iov\_base*.13484 The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.13485 The **<sys/uio.h>** header shall define the **ssize\_t** and **size\_t** types as described in **<sys/types.h>**.13486 The following shall be declared as functions and may also be defined as macros. Function  
13487 prototypes shall be provided.13488 `ssize_t readv(int, const struct iovec *, int);`  
13489 `ssize_t writev(int, const struct iovec *, int);`13490 **APPLICATION USAGE**13491 The implementation can put a limit on the number of scatter/gather elements which can be  
13492 processed in one call. The symbol {IOV\_MAX} defined in **<limits.h>** should always be used to  
13493 learn about the limits instead of assuming a fixed value.13494 **RATIONALE**13495 Traditionally, the maximum number of scatter/gather elements the system can process in one  
13496 call were described by the symbolic value {UIO\_MAXIOV}. In IEEE Std 1003.1-2001 this value is  
13497 replaced by the constant {IOV\_MAX} which can be found in **<limits.h>**.13498 **FUTURE DIRECTIONS**

13499 None.

13500 **SEE ALSO**13501 **<limits.h>**, **<sys/types.h>**13502 XSH *read()*, *readv()*, *write()*, *writev()*13503 **CHANGE HISTORY**

13504 First released in Issue 4, Version 2.

13505 **Issue 6**

13506 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13507 **NAME**

13508 sys/un.h — definitions for UNIX domain sockets

13509 **SYNOPSIS**

13510 #include &lt;sys/un.h&gt;

13511 **DESCRIPTION**13512 The <sys/un.h> header shall define the **sockaddr\_un** structure, which shall include at least the  
13513 following members:13514 sa\_family\_t sun\_family Address family.  
13515 char sun\_path[] Socket pathname.13516 The **sockaddr\_un** structure is used to store addresses for UNIX domain sockets. Values of this  
13517 type shall be cast by applications to **struct sockaddr** for use with socket functions.13518 The <sys/un.h> header shall define the **sa\_family\_t** type as described in <sys/socket.h>.13519 **APPLICATION USAGE**13520 The size of *sun\_path* has intentionally been left undefined. This is because different  
13521 implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a  
13522 size of 104. Since most implementations originate from BSD versions, the size is typically in the  
13523 range 92 to 108.13524 Applications should not assume a particular length for *sun\_path* or assume that it can hold  
13525 {\_POSIX\_PATH\_MAX} bytes (256).13526 **RATIONALE**

13527 None.

13528 **FUTURE DIRECTIONS**

13529 None.

13530 **SEE ALSO**

13531 &lt;sys/socket.h&gt;

13532 XSH *bind()*, *socket()*, *socketpair()*13533 **CHANGE HISTORY**

13534 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13535 **Issue 7**

13536 The value for {\_POSIX\_PATH\_MAX} is updated to 256.

## &lt;sys/utsname.h&gt;

Headers

13537 **NAME**

13538 sys/utsname.h — system name structure

13539 **SYNOPSIS**

13540 #include &lt;sys/utsname.h&gt;

13541 **DESCRIPTION**13542 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the  
13543 following members:

13544 char sysname[] Name of this implementation of the operating system.  
 13545 char nodename[] Name of this node within the communications  
 13546 network to which this node is attached, if any.  
 13547 char release[] Current release level of this implementation.  
 13548 char version[] Current version level of this release.  
 13549 char machine[] Name of the hardware type on which the system is running.

13550 The character arrays are of unspecified size, but the data stored in them shall be terminated by a  
 13551 null byte.

13552 The following shall be declared as a function and may also be defined as a macro:

13553 int uname(struct utsname \*);

13554 **APPLICATION USAGE**

13555 None.

13556 **RATIONALE**

13557 None.

13558 **FUTURE DIRECTIONS**

13559 None.

13560 **SEE ALSO**13561 XSH *uname()*13562 **CHANGE HISTORY**

13563 First released in Issue 1. Derived from Issue 1 of the SVID.

13564 **Issue 6**

13565 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of  
 13566 *nodename* within the **utsname** structure from “an implementation-defined communications  
 13567 network” to “the communications network to which this node is attached, if any”.

## Headers

## &lt;sys/wait.h&gt;

## 13568 NAME

13569 sys/wait.h — declarations for waiting

## 13570 SYNOPSIS

13571 #include <sys/wait.h>

## 13572 DESCRIPTION

13573 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13574 XSI **WCONTINUED** Report status of continued child process.  
 13575 **WNOHANG** Do not hang if no status is available; return immediately.  
 13576 **WUNTRACED** Report status of stopped child process.

13577 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13578 **WEXITSTATUS** Return exit status.  
 13579 XSI **WIFCONTINUED** True if child has been continued.  
 13580 **WIFEXITED** True if child exited normally.  
 13581 **WIFSIGNALED** True if child exited due to uncaught signal.  
 13582 **WIFSTOPPED** True if child is currently stopped.  
 13583 **WSTOPSIG** Return signal number that caused process to stop.  
 13584 **WTERMSIG** Return signal number that caused process to terminate.

13585 The <sys/wait.h> header shall define the following symbolic constants as possible values for the  
 13586 *options* argument to *waitid()*:

13587 **WEXITED** Wait for processes that have exited.  
 13588 **WNOWAIT** Keep the process whose status is returned in *infp* in a waitable state.  
 13589 **WSTOPPED** Status is returned for any child that has stopped upon receipt of a signal.

13590 XSI The **WCONTINUED** and **WNOHANG** constants, described above for *waitpid()*, can also be  
 13591 used with *waitid()*.

13592 The type **idtype\_t** shall be defined as an enumeration type whose possible values shall include  
 13593 at least the following:

13594 P\_ALL  
 13595 P\_PGID  
 13596 P\_PID

13597 The <sys/wait.h> header shall define the **id\_t** and **pid\_t** types as described in <sys/types.h>.

13598 The <sys/wait.h> header shall define the **siginfo\_t** type as described in <signal.h>.

13599 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h>.

13600 The following shall be declared as functions and may also be defined as macros. Function  
 13601 prototypes shall be provided.

```
13602 pid_t wait(int *);
13603 int waitid(idtype_t, id_t, siginfo_t *, int);
13604 pid_t waitpid(pid_t, int *, int);
```

## &lt;sys/wait.h&gt;

Headers

13605 **APPLICATION USAGE**

13606 None.

13607 **RATIONALE**

13608 None.

13609 **FUTURE DIRECTIONS**

13610 None.

13611 **SEE ALSO**13612 [<signal.h>](#), [<sys/types.h>](#)13613 XSH *wait()*, *waitid()*13614 **CHANGE HISTORY**

13615 First released in Issue 3.

13616 Included for alignment with the POSIX.1-1988 standard.

13617 **Issue 6**13618 The *wait3()* function is removed.13619 **Issue 7**13620 The *waitid()* function and symbolic constants for its *options* argument are moved to the Base.

13621 The description of the WNOHANG constant is clarified.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

13622 **NAME**

13623            syslog.h — definitions for system error logging

13624 **SYNOPSIS**

13625 XSI        #include &lt;syslog.h&gt;

13626 **DESCRIPTION**13627            The <syslog.h> header shall define the following symbolic constants, zero or more of which  
13628            may be OR'ed together to form the *logopt* option of *openlog()*:

13629            LOG\_PID                Log the process ID with each message.

13630            LOG\_CONS               Log to the system console on error.

13631            LOG\_NDELAY             Connect to syslog daemon immediately.

13632            LOG\_ODELAY             Delay open until *syslog()* is called.

13633            LOG\_NOWAIT            Do not wait for child processes.

13634            The <syslog.h> header shall define the following symbolic constants for use as the *facility*  
13635            argument to *openlog()*:

13636            LOG\_KERN               Reserved for message generated by the system.

13637            LOG\_USER               Message generated by a process.

13638            LOG\_MAIL               Reserved for message generated by mail system.

13639            LOG\_NEWS               Reserved for message generated by news system.

13640            LOG\_UUCP               Reserved for message generated by UUCP system.

13641            LOG\_DAEMON            Reserved for message generated by system daemon.

13642            LOG\_AUTH               Reserved for message generated by authorization daemon.

13643            LOG\_CRON               Reserved for message generated by clock daemon.

13644            LOG\_LPR                Reserved for message generated by printer system.

13645            LOG\_LOCAL0            Reserved for local use.

13646            LOG\_LOCAL1            Reserved for local use.

13647            LOG\_LOCAL2            Reserved for local use.

13648            LOG\_LOCAL3            Reserved for local use.

13649            LOG\_LOCAL4            Reserved for local use.

13650            LOG\_LOCAL5            Reserved for local use.

13651            LOG\_LOCAL6            Reserved for local use.

13652            LOG\_LOCAL7            Reserved for local use.

13653            The <syslog.h> header shall define the following macros for constructing the *maskpri* argument  
13654            to *setlogmask()*. The following macros expand to an expression of type **int** when the argument  
13655            *pri* is an expression of type **int**:13656            LOG\_MASK(*pri*)        A mask for priority *pri*.13657            The <syslog.h> header shall define the following symbolic constants for use as the *priority*  
13658            argument of *syslog()*:

## &lt;syslog.h&gt;

Headers

13659	LOG_EMERG	A panic condition was reported to all processes.
13660	LOG_ALERT	A condition that should be corrected immediately.
13661	LOG_CRIT	A critical condition.
13662	LOG_ERR	An error message.
13663	LOG_WARNING	A warning message.
13664	LOG_NOTICE	A condition requiring special handling.
13665	LOG_INFO	A general information message.
13666	LOG_DEBUG	A message useful for debugging programs.
13667	The following shall be declared as functions and may also be defined as macros. Function	
13668	prototypes shall be provided.	
13669	void	closelog(void);
13670	void	openlog(const char *, int, int);
13671	int	setlogmask(int);
13672	void	syslog(int, const char *, ...);
13673	<b>APPLICATION USAGE</b>	
13674	None.	
13675	<b>RATIONALE</b>	
13676	None.	
13677	<b>FUTURE DIRECTIONS</b>	
13678	None.	
13679	<b>SEE ALSO</b>	
13680	XSH	<i>closelog()</i>
13681	<b>CHANGE HISTORY</b>	
13682	First released in Issue 4, Version 2.	
13683	<b>Issue 5</b>	
13684	Moved from X/Open UNIX to BASE.	
13685	<b>Issue 7</b>	
13686	This reference page is clarified with respect to macros and symbolic constants.	

## Headers

## &lt;tar.h&gt;

13687 **NAME**

13688 tar.h — extended tar definitions

13689 **SYNOPSIS**

13690 #include &lt;tar.h&gt;

13691 **DESCRIPTION**

13692 The &lt;tar.h&gt; header shall define the following symbolic constants with the indicated values.

13693 General definitions:

Name	Value	Description
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13699 *Typeflag* field definitions:

Name	Value	Description
REGTYPE	' 0'	Regular file.
AREGTYPE	' \0'	Regular file.
LNKTYPE	' 1'	Link.
SYMTYPE	' 2'	Symbolic link.
CHRTYPE	' 3'	Character special.
BLKTYPE	' 4'	Block special.
DIRTYPE	' 5'	Directory.
FIFOTYPE	' 6'	FIFO special.
CONTTYPE	' 7'	Reserved.

13710 *Mode* field bit definitions (octal):

Name	Value	Description
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

## &lt;tar.h&gt;

Headers

- 13724 **APPLICATION USAGE**  
13725           None.
- 13726 **RATIONALE**  
13727           None.
- 13728 **FUTURE DIRECTIONS**  
13729           None.
- 13730 **SEE ALSO**  
13731           XCU *pax*
- 13732 **CHANGE HISTORY**  
13733           First released in Issue 3. Derived from the POSIX.1-1988 standard.
- 13734 **Issue 6**  
13735           The SEE ALSO section is updated to refer to *pax*.
- 13736 **Issue 7**  
13737           This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

13738 **NAME**13739 `termios.h` — define values for `termios`13740 **SYNOPSIS**13741 `#include <termios.h>`13742 **DESCRIPTION**13743 The `<termios.h>` header shall contain the definitions used by the terminal I/O interfaces (see  
13744 [Chapter 11](#) (on page 199) for the structures and names defined).13745 **The `termios` Structure**13746 The `<termios.h>` header shall define the following data types through **typedef**:13747 **cc\_t** Used for terminal special characters.13748 **speed\_t** Used for terminal baud rates.13749 **tcflag\_t** Used for terminal modes.

13750 The above types shall be all unsigned integer types.

13751 The implementation shall support one or more programming environments in which the widths  
13752 of **cc\_t**, **speed\_t**, and **tcflag\_t** are no greater than the width of type **long**. The names of these  
13753 programming environments can be obtained using the `confstr()` function or the `getconf` utility.13754 The `<termios.h>` header shall define the **termios** structure, which shall include at least the  
13755 following members:

13756	<code>tcflag_t</code>	<code>c_iflag</code>	Input modes.
13757	<code>tcflag_t</code>	<code>c_oflag</code>	Output modes.
13758	<code>tcflag_t</code>	<code>c_cflag</code>	Control modes.
13759	<code>tcflag_t</code>	<code>c_lflag</code>	Local modes.
13760	<code>cc_t</code>	<code>c_cc[NCCS]</code>	Control characters.

13761 The `<termios.h>` header shall define the following symbolic constant:13762 **NCCS** Size of the array `c_cc` for control characters.13763 The `<termios.h>` header shall define the following symbolic constants for use as subscripts for  
13764 the array `c_cc`:

	Subscript Usage		Description
	Canonical Mode	Non-Canonical Mode	
13765	VEOF		EOF character.
13766	VEOL		EOL character.
13767	VERASE		ERASE character.
13768	VINTR	VINTR	INTR character.
13769	VKILL		KILL character.
13770		VMIN	MIN value.
13771	VQUIT	VQUIT	QUIT character.
13772	VSTART	VSTART	START character.
13773	VSTOP	VSTOP	STOP character.
13774	VSUSP	VSUSP	SUSP character.
13775		VTIME	TIME value.

13778 The subscript values shall be suitable for use in `#if` preprocessing directives and shall be distinct,  
13779 except that the **VMIN** and **VTIME** subscripts may have the same values as the **VEOF** and **VEOL**  
13780 subscripts, respectively.

<termios.h>

13781 **Input Modes**

13782 The <termios.h> header shall define the following symbolic constants for use as flags in the  
13783 *c\_iflag* field. The *c\_iflag* field describes the basic terminal input control.

- 13784 BRKINT Signal interrupt on break.
- 13785 ICRNL Map CR to NL on input.
- 13786 IGNBRK Ignore break condition.
- 13787 IGNCR Ignore CR.
- 13788 IGNPAR Ignore characters with parity errors.
- 13789 INLCR Map NL to CR on input.
- 13790 INPCK Enable input parity check.
- 13791 ISTRIP Strip character.
- 13792 IXANY Enable any character to restart output.
- 13793 IXOFF Enable start/stop input control.
- 13794 IXON Enable start/stop output control.
- 13795 PARMRK Mark parity errors.

13796 **Output Modes**

13797 The <termios.h> header shall define the following symbolic constants for use as flags in the  
13798 *c\_oflag* field. The *c\_oflag* field specifies the system treatment of output.

- 13799 OPOST Post-process output.
- 13800 XSI ONLCR Map NL to CR-NL on output.
- 13801 XSI OCRNL Map CR to NL on output.
- 13802 XSI ONOCR No CR output at column 0.
- 13803 XSI ONLRET NL performs CR function.
- 13804 XSI OFDEL Fill is DEL.
- 13805 XSI OFILL Use fill characters for delay.
- 13806 XSI NLDLY Select newline delays:
  - 13807 NL0 Newline type 0.
  - 13808 NL1 Newline type 1.
- 13809 XSI CRDLY Select carriage-return delays:
  - 13810 CR0 Carriage-return delay type 0.
  - 13811 CR1 Carriage-return delay type 1.
  - 13812 CR2 Carriage-return delay type 2.
  - 13813 CR3 Carriage-return delay type 3.

## Headers

## &lt;termios.h&gt;

13814	XSI	<b>TABDLY</b>	Select horizontal-tab delays:
13815		TAB0	Horizontal-tab delay type 0.
13816		TAB1	Horizontal-tab delay type 1.
13817		TAB2	Horizontal-tab delay type 2.
13818		TAB3	Expand tabs to spaces.
13819	XSI	<b>BSDLY</b>	Select backspace delays:
13820		BS0	Backspace-delay type 0.
13821		BS1	Backspace-delay type 1.
13822	XSI	<b>VTDLY</b>	Select vertical-tab delays:
13823		VT0	Vertical-tab delay type 0.
13824		VT1	Vertical-tab delay type 1.
13825	XSI	<b>FFDLY</b>	Select form-feed delays:
13826		FF0	Form-feed delay type 0.
13827		FF1	Form-feed delay type 1.

13828 **Baud Rate Selection**

13829 The <termios.h> header shall define the following symbolic constants for use as values of  
 13830 objects of type **speed\_t**.

13831 The input and output baud rates are stored in the **termios** structure. These are the valid values  
 13832 for objects of type **speed\_t**. Not all baud rates need be supported by the underlying hardware.

13833	B0	Hang up
13834	B50	50 baud
13835	B75	75 baud
13836	B110	110 baud
13837	B134	134.5 baud
13838	B150	150 baud
13839	B200	200 baud
13840	B300	300 baud
13841	B600	600 baud
13842	B1200	1 200 baud
13843	B1800	1 800 baud
13844	B2400	2 400 baud

## &lt;termios.h&gt;

## Headers

13845	B4800	4 800 baud
13846	B9600	9 600 baud
13847	B19200	19 200 baud
13848	B38400	38 400 baud

13849 **Control Modes**

13850 The <termios.h> header shall define the following symbolic constants for use as flags in the  
 13851 *c\_cflag* field. The *c\_cflag* field describes the hardware control of the terminal; not all values  
 13852 specified are required to be supported by the underlying hardware.

13853	CSIZE	Character size:
13854		CS5 5 bits
13855		CS6 6 bits
13856		CS7 7 bits
13857		CS8 8 bits
13858	CSTOPB	Send two stop bits, else one.
13859	CREAD	Enable receiver.
13860	PARENB	Parity enable.
13861	PARODD	Odd parity, else even.
13862	HUPCL	Hang up on last close.
13863	CLOCAL	Ignore modem status lines.

13864 The implementation shall support the functionality associated with the symbols CS7, CS8,  
 13865 CSTOPB, PARODD, and PARENB.

13866 **Local Modes**

13867 The <termios.h> header shall define the following symbolic constants for use as flags in the  
 13868 *c\_lflag* field. The *c\_lflag* field of the argument structure is used to control various terminal  
 13869 functions.

13870	ECHO	Enable echo.
13871	ECHOE	Echo erase character as error-correcting backspace.
13872	ECHOK	Echo KILL.
13873	ECHONL	Echo NL.
13874	ICANON	Canonical input (erase and kill processing).
13875	IEXTEN	Enable extended input character processing.
13876	ISIG	Enable signals.
13877	NOFLSH	Disable flush after interrupt or quit.
13878	TOSTOP	Send SIGTTOU for background output.

13879 **Attribute Selection**

13880 The <termios.h> header shall define the following symbolic constants for use with *tcsetattr()*:

- 13881 TCSANOW Change attributes immediately.
- 13882 TCSADRAIN Change attributes when output has drained.
- 13883 TCSAFLUSH Change attributes when output has drained; also flush pending input.

13884 **Line Control**

13885 The <termios.h> header shall define the following symbolic constants for use with *tcflush()*:

- 13886 TCIFLUSH Flush pending input.
- 13887 TCIOFLUSH Flush both pending input and untransmitted output.
- 13888 TCOFLUSH Flush untransmitted output.

13889 The <termios.h> header shall define the following symbolic constants for use with *tcflow()*:

- 13890 TCIOFF Transmit a STOP character, intended to suspend input data.
- 13891 TCION Transmit a START character, intended to restart input data.
- 13892 TCOOFF Suspend output.
- 13893 TCOON Restart output.

13894 The <termios.h> header shall define the **pid\_t** type as described in <sys/types.h>.

13895 The following shall be declared as functions and may also be defined as macros. Function  
13896 prototypes shall be provided.

```
13897 speed_t cfgetispeed(const struct termios *);
13898 speed_t cfgetospeed(const struct termios *);
13899 int cfsetispeed(struct termios *, speed_t);
13900 int cfsetospeed(struct termios *, speed_t);
13901 int tcdrain(int);
13902 int tcflow(int, int);
13903 int tcflush(int, int);
13904 int tcgetattr(int, struct termios *);
13905 pid_t tcgetsid(int);
13906 int tcsendbreak(int, int);
13907 int tcsetattr(int, int, const struct termios *);
```

13908 **APPLICATION USAGE**

13909 The following names are reserved for XSI-conformant systems to use as an extension to the  
13910 above; therefore strictly conforming applications shall not use them:

13911	CBAUD	EXTB	VDSUSP
13912	DEFECHO	FLUSHO	VLNEXT
13913	ECHOCTL	LOBLK	VREPRINT
13914	ECHOKE	PENDIN	VSTATUS
13915	ECHOPRT	SWTCH	VWERASE
13916	EXTA	VDISCARD	

## &lt;termios.h&gt;

Headers

- 13917 **RATIONALE**  
 13918 None.
- 13919 **FUTURE DIRECTIONS**  
 13920 None.
- 13921 **SEE ALSO**  
 13922 [<sys/types.h>](#)  
 13923 XSH [cfgetispeed\(\)](#), [cfgetospeed\(\)](#), [cfsetispeed\(\)](#), [cfsetospeed\(\)](#), [confstr\(\)](#), [tcdrain\(\)](#), [tcflow\(\)](#), [tcflush\(\)](#),  
 13924 [tcgetattr\(\)](#), [tcgetsid\(\)](#), [tcsendbreak\(\)](#), [tcsetattr\(\)](#)  
 13925 XCU [Chapter 11](#) (on page 199), [getconf](#)
- 13926 **CHANGE HISTORY**  
 13927 First released in Issue 3.  
 13928 Included for alignment with the ISO POSIX-1 standard.
- 13929 **Issue 6**  
 13930 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.  
 13931 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are  
 13932 reaffirmed.  
 13933 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to  
 13934 ECHOKE in the APPLICATION USAGE section.
- 13935 **Issue 7**  
 13936 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the  
 13937 IXANY symbol from the XSI option to the Base.  
 13938 SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.  
 13939 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
 13940 for the `pid_t` type is added.

## Headers

## &lt;tgmath.h&gt;

## 13941 NAME

13942 tgmath.h — type-generic macros

## 13943 SYNOPSIS

13944 #include &lt;tgmath.h&gt;

## 13945 DESCRIPTION

13946 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13947 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13948 volume of POSIX.1-2008 defers to the ISO C standard.

13949 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define  
 13950 several type-generic macros.

13951 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**)  
 13952 or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is  
 13953 XSI **double**. For each such function, except *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, there shall  
 13954 be a corresponding type-generic macro. The parameters whose corresponding real type is  
 13955 **double** in the function synopsis are generic parameters. Use of the macro invokes a function  
 13956 whose corresponding real type and type domain are determined by the arguments for the  
 13957 generic parameters.

13958 Use of the macro invokes a function whose generic parameters have the corresponding real type  
 13959 determined as follows:

- 13960 • First, if any argument for generic parameters has type **long double**, the type determined is  
 13961 **long double**.
- 13962 • Otherwise, if any argument for generic parameters has type **double** or is of integer type,  
 13963 the type determined is **double**.
- 13964 • Otherwise, the type determined is **float**.

13965 For each unsuffixed function in the <math.h> header for which there is a function in the  
 13966 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic  
 13967 macro (for both functions) has the same name as the function in the <math.h> header. The  
 13968 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

13969

13970

13971

13972

13973

13974

13975

13976

13977

13978

13979

13980

13981

13982

13983

13984

13985

13986

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

<tgmath.h>

13987 If at least one argument for a generic parameter is complex, then use of the macro invokes a  
 13988 complex function; otherwise, use of the macro invokes a real function.

13989 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the  
 13990 XSI <complex.h> header, except for *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, the corresponding  
 13991 type-generic macro has the same name as the function. These type-generic macros are:

13992	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
13993	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
13994	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
13995	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
13996	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
13997	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
13998	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
13999	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
14000	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
14001	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

14002 If all arguments for generic parameters are real, then use of the macro invokes a real function;  
 14003 otherwise, use of the macro results in undefined behavior.

14004 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a  
 14005 function in the <math.h> header, the corresponding type-generic macro has the same name as  
 14006 the function. These type-generic macros are:

14007	<i>carg()</i>
14008	<i>cimag()</i>
14009	<i>conj()</i>
14010	<i>cproj()</i>
14011	<i>creal()</i>

14012 Use of the macro with any real or complex argument invokes a complex function.

**APPLICATION USAGE**

14013 With the declarations:

```
14014 #include <tgmath.h>
14015 int n;
14016 float f;
14017 double d;
14018 long double ld;
14019 float complex fc;
14020 double complex dc;
14021 long double complex ldc;
```

14022 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acosh(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>
<i>log(fc)</i>	<i>clog(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(dc)</i>

	Macro	Use Invokes
14031		
14032	<i>pow(ldc,f)</i>	<i>cpowl(ldc,f)</i>
14033	<i>remainder(n,n)</i>	<i>remainder(n,n)</i> , the function
14034	<i>nextafter(d,f)</i>	<i>nextafter(d,f)</i> , the function
14035	<i>nexttoward(f,ld)</i>	<i>nexttowardf(f,ld)</i>
14036	<i>copysign(n,ld)</i>	<i>copysignl(n,ld)</i>
14037	<i>ceil(fc)</i>	Undefined behavior
14038	<i>rint(dc)</i>	Undefined behavior
14039	<i>fmax(ldc,ld)</i>	Undefined behavior
14040	<i>carg(n)</i>	<i>carg(n)</i> , the function
14041	<i>cproj(f)</i>	<i>cprojf(f)</i>
14042	<i>creal(d)</i>	<i>creal(d)</i> , the function
14043	<i>cimag(ld)</i>	<i>cimagl(ld)</i>
14044	<i>cabs(fc)</i>	<i>cabsf(fc)</i>
14045	<i>carg(dc)</i>	<i>carg(dc)</i> , the function
14046	<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

14047 **RATIONALE**

14048 Type-generic macros allow calling a function whose type is determined by the argument type, as  
 14049 is the case for C operators such as '+' and '\*'. For example, with a type-generic *cos()* macro,  
 14050 the expression *cos((float)x)* will have type **float**. This feature enables writing more portably  
 14051 efficient code and alleviates need for awkward casting and suffixing in the process of porting or  
 14052 adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

14053 The only arguments that affect the type resolution are the arguments corresponding to the  
 14054 parameters that have type **double** in the synopsis. Hence the type of a type-generic call to  
 14055 *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by  
 14056 the type of the first argument.

14057 The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading.  
 14058 The term is more specific than intrinsic, which already is widely used with a more general  
 14059 meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

14060 The macros are placed in their own header in order not to silently break old programs that  
 14061 include the <math.h> header; for example, with:

```
14062 printf ("%e", sin(x))
```

14063 *modf(double, double \*)* is excluded because no way was seen to make it safe without  
 14064 complicating the type resolution.

14065 The implementation might, as an extension, endow appropriate ones of the macros that  
 14066 POSIX.1-2008 specifies only for real arguments with the ability to invoke the complex functions.

14067 POSIX.1-2008 does not prescribe any particular implementation mechanism for generic macros.  
 14068 It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example,  
 14069 could be implemented with:

```
14070 #undef sqrt
14071 #define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

14072 Generic macros are designed for a useful level of consistency with C++ overloaded math  
 14073 functions.

14074 The great majority of existing C programs are expected to be unaffected when the <tgmath.h>  
 14075 header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar  
 14076 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return

**<tgmath.h>**

Headers

- 14077 values differ.
- 14078 The ability to overload on integer as well as floating types would have been useful for some  
14079 functions; for example, *copysign()*. Overloading with different numbers of arguments would  
14080 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities  
14081 would have complicated the specification; and their natural consistent use, such as for a floating  
14082 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the  
14083 ISO/IEC 9899:1999 standard for insufficient benefit.
- 14084 The ISO C standard in no way limits the implementation's options for efficiency, including  
14085 inlining library functions.
- 14086 **FUTURE DIRECTIONS**
- 14087 None.
- 14088 **SEE ALSO**
- 14089 [<math.h>](#), [<complex.h>](#)
- 14090 XSH *cabs()*, *fabs()*, *modf()*
- 14091 **CHANGE HISTORY**
- 14092 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.
- 14093 **Issue 7**
- 14094 Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a  
14095 corresponding type-generic macro exists with the same name as the function.

## Headers

## &lt;time.h&gt;

## 14096 NAME

14097 time.h — time types

## 14098 SYNOPSIS

14099 #include &lt;time.h&gt;

## 14100 DESCRIPTION

14101 CX Some of the functionality described on this reference page extends the ISO C standard.  
 14102 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 14103 enable the visibility of these symbols in this header.

14104 The <time.h> header shall define the **clock\_t**, **size\_t**, **time\_t**, types as described in  
 14105 <sys/types.h>.

14106 CX The <time.h> header shall define the **clockid\_t** and **timer\_t** types as described in <sys/types.h>.

14107 CX The <time.h> header shall define the **locale\_t** type as described in <locale.h>.

14108 CPT The <time.h> header shall define the **pid\_t** type as described in <sys/types.h>.

14109 CX The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which  
 14110 are described in the <signal.h> header.

14111 The <time.h> header shall declare the **tm** structure, which shall include at least the following  
 14112 members:

14113	int	tm_sec	Seconds [0,60].
14114	int	tm_min	Minutes [0,59].
14115	int	tm_hour	Hour [0,23].
14116	int	tm_mday	Day of month [1,31].
14117	int	tm_mon	Month of year [0,11].
14118	int	tm_year	Years since 1900.
14119	int	tm_wday	Day of week [0,6] (Sunday =0).
14120	int	tm_yday	Day of year [0,365].
14121	int	tm_isdst	Daylight Savings flag.

14122 The value of *tm\_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings  
 14123 Time is not in effect, and negative if the information is not available.

14124 CX The <time.h> header shall declare the **timespec** structure, which shall include at least the  
 14125 following members:

14126	time_t	tv_sec	Seconds.
14127	long	tv_nsec	Nanoseconds.

14128 The <time.h> header shall also declare the **itimerspec** structure, which shall include at least the  
 14129 following members:

14130	struct timespec	it_interval	Timer period.
14131	struct timespec	it_value	Timer expiration.

14132 The <time.h> header shall define the following macros:

14133 NULL As described in <stddef.h>.

14134 CLOCKS\_PER\_SEC A number used to convert the value returned by the *clock()* function into  
 14135 XSI seconds. The value shall be an expression with type **clock\_t**. The value of  
 14136 CLOCKS\_PER\_SEC shall be 1 million on XSI-conformant systems.

14137		However, it may be variable on other systems, and it should not be
14138		assumed that <code>CLOCKS_PER_SEC</code> is a compile-time constant.
14139	CX	The <time.h> header shall define the following symbolic constants. The values shall have a type
14140		that is assignment-compatible with <code>clockid_t</code> .
14141	MON	<code>CLOCK_MONOTONIC</code>
14142		The identifier for the system-wide monotonic clock, which is defined as a
14143		clock measuring real time, whose value cannot be set via <code>clock_settime()</code>
14144		and which cannot have negative clock jumps. The maximum possible
14145		clock jump shall be implementation-defined.
14146	CPT	<code>CLOCK_PROCESS_CPUTIME_ID</code>
14147		The identifier of the CPU-time clock associated with the process making a
14148		<code>clock()</code> or <code>timer*()</code> function call.
14149	CX	<code>CLOCK_REALTIME</code> The identifier of the system-wide clock measuring real time.
14150	TCT	<code>CLOCK_THREAD_CPUTIME_ID</code>
14151		The identifier of the CPU-time clock associated with the thread making a
14152		<code>clock()</code> or <code>timer*()</code> function call.
14153		The <time.h> header shall define the following symbolic constant:
14154		<code>TIMER_ABSTIME</code> Flag indicating time is absolute. For functions taking timer objects, this
14155		refers to the clock associated with the timer.
14156	XSI	The <time.h> header shall provide a declaration or definition for <code>getdate_err</code> . The <code>getdate_err</code>
14157		symbol shall expand to an expression of type <code>int</code> . It is unspecified whether <code>getdate_err</code> is a macro
14158		or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a
14159		macro definition is suppressed in order to access an actual object, or a program defines an
14160		identifier with the name <code>getdate_err</code> , the behavior is undefined.
14161		The following shall be declared as functions and may also be defined as macros. Function
14162		prototypes shall be provided.
14163	OB	<code>char *asctime(const struct tm *);</code>
14164	OB CX	<code>char *asctime_r(const struct tm *restrict, char *restrict);</code>
14165		<code>clock_t clock(void);</code>
14166	CPT	<code>int clock_getcpuclockid(pid_t, clockid_t *);</code>
14167	CX	<code>int clock_getres(clockid_t, struct timespec *);</code>
14168		<code>int clock_gettime(clockid_t, struct timespec *);</code>
14169		<code>int clock_nanosleep(clockid_t, int, const struct timespec *,</code>
14170		<code>struct timespec *);</code>
14171		<code>int clock_settime(clockid_t, const struct timespec *);</code>
14172	OB	<code>char *ctime(const time_t *);</code>
14173	OB CX	<code>char *ctime_r(const time_t *, char *);</code>
14174		<code>double difftime(time_t, time_t);</code>
14175	XSI	<code>struct tm *getdate(const char *);</code>
14176		<code>struct tm *gmtime(const time_t *);</code>
14177	CX	<code>struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);</code>
14178		<code>struct tm *localtime(const time_t *);</code>
14179	CX	<code>struct tm *localtime_r(const time_t *restrict, struct tm *restrict);</code>
14180		<code>time_t mktime(struct tm *);</code>
14181	CX	<code>int nanosleep(const struct timespec *, struct timespec *);</code>
14182		<code>size_t strftime(char *restrict, size_t, const char *restrict,</code>
14183		<code>const struct tm *restrict);</code>

```

14184 CX      size_t      strftime_l(char *restrict, size_t, const char *restrict,
14185           const struct tm *restrict, locale_t);
14186 XSI      char        *strptime(const char *restrict, const char *restrict,
14187           struct tm *restrict);
14188           time_t      time(time_t *);
14189 CX      int          timer_create(clockid_t, struct sigevent *restrict,
14190           timer_t *restrict);
14191           int          timer_delete(timer_t);
14192           int          timer_getoverrun(timer_t);
14193           int          timer_gettime(timer_t, struct itimerspec *);
14194           int          timer_settime(timer_t, int, const struct itimerspec *restrict,
14195           struct itimerspec *restrict);
14196           void         tzset(void);

```

14197 The <time.h> header shall declare the following as variables:

```

14198 XSI      extern int     daylight;
14199           extern long    timezone;
14200 CX      extern char    *tzname[];

```

14201 CX Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.

#### 14202 APPLICATION USAGE

14203 The range [0,60] for *tm\_sec* allows for the occasional leap second.

14204 *tm\_year* is a signed value; therefore, years before 1900 may be represented.

14205 To obtain the number of clock ticks per second returned by the *times()* function, applications  
14206 should call *sysconf(\_SC\_CLK\_TCK)*.

#### 14207 RATIONALE

14208 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of  
14209 UTC does not permit double leap seconds, so all mention of double leap seconds has been  
14210 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this  
14211 standard.

#### 14212 FUTURE DIRECTIONS

14213 None.

#### 14214 SEE ALSO

14215 <locale.h>, <signal.h>, <stddef.h>, <sys/types.h>

14216 XSH Section 2.2 (on page 468), *asctime()*, *clock()*, *clock\_getcpuclockid()*, *clock\_getres()*,  
14217 *clock\_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *mq\_receive()*,  
14218 *mq\_send()*, *nanosleep()*, *pthread\_getcpuclockid()*, *pthread\_mutex\_timedlock()*,  
14219 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *sem\_timedwait()*, *strftime()*, *strptime()*,  
14220 *sysconf()*, *time()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *tzset()*, *utime()*

#### 14221 CHANGE HISTORY

14222 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 14223 Issue 5

14224 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
14225 Threads Extension.

14226 **Issue 6**

14227 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid\_t**  
 14228 and **timer\_t** have been described.

14229 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 14230 • The POSIX timer-related functions are marked as part of the Timers option.

14231 The symbolic name CLK\_TCK is removed. Application usage is added describing how its  
 14232 equivalent functionality can be obtained using *sysconf()*.

14233 The *clock\_getcpuclockid()* function and manifest constants CLOCK\_PROCESS\_CPUTIME\_ID and  
 14234 CLOCK\_THREAD\_CPUTIME\_ID are added for alignment with IEEE Std 1003.1d-1999.

14235 The manifest constant CLOCK\_MONOTONIC and the *clock\_nanosleep()* function are added for  
 14236 alignment with IEEE Std 1003.1j-2000.

14237 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 14238 • The range for seconds is changed from [0,61] to [0,60].
- 14239 • The **restrict** keyword is added to the prototypes for *asctime\_r()*, *gmtime\_r()*, *localtime\_r()*,  
 14240 *strftime()*, *strptime()*, *timer\_create()*, and *timer\_settime()*.

14241 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the  
 14242 <**signal.h**> header may be made visible when the <**time.h**> header is included.

14243 Extensions beyond the ISO C standard are marked.

14244 **Issue 7**

14245 Austin Group Interpretation 1003.1-2001 #111 is applied.

14246 SD5-XBD-ERN-74 is applied.

14247 The *strftime\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
 14248 Set Part 4.

14249 Functionality relating to the Timers option is moved to the Base.

14250 This reference page is clarified with respect to macros and symbolic constants, and declarations  
 14251 for the **locale\_t** and **pid\_t** types and the **sigevent** structure are added.

14252 The description of the *getdate\_err* value is expanded.

## Headers

## &lt;trace.h&gt;

## 14253 NAME

14254 trace.h — tracing

## 14255 SYNOPSIS

14256 OB TRC #include &lt;trace.h&gt;

## 14257 DESCRIPTION

14258 The <trace.h> header shall define the **posix\_trace\_event\_info** structure, which shall include at  
14259 least the following members:

```

14260 trace_event_id_t  posix_event_id
14261 pid_t             posix_pid
14262 void              *posix_prog_address
14263 pthread_t         posix_thread_id
14264 struct timespec   posix_timestamp
14265 int               posix_truncation_status

```

14266 The <trace.h> header shall define the **posix\_trace\_status\_info** structure, which shall include at  
14267 least the following members:

```

14268 int      posix_stream_full_status
14269 int      posix_stream_overrun_status
14270 int      posix_stream_status
14271 OB TRL  int      posix_log_full_status
14272 int      posix_log_overrun_status
14273 int      posix_stream_flush_error
14274 int      posix_stream_flush_status

```

14275 The &lt;trace.h&gt; header shall define the following symbolic constants:

```

14276 POSIX_TRACE_ALL_EVENTS
14277 OB TRL  POSIX_TRACE_APPEND
14278 OB TRI  POSIX_TRACE_CLOSE_FOR_CHILD
14279 OB TEF  POSIX_TRACE_FILTER
14280 OB TRL  POSIX_TRACE_FLUSH
14281        POSIX_TRACE_FLUSH_START
14282        POSIX_TRACE_FLUSH_STOP
14283        POSIX_TRACE_FLUSHING
14284        POSIX_TRACE_FULL
14285        POSIX_TRACE_LOOP
14286        POSIX_TRACE_NO_OVERRUN
14287 OB TRL  POSIX_TRACE_NOT_FLUSHING
14288        POSIX_TRACE_NOT_FULL
14289 OB TRI  POSIX_TRACE_INHERITED
14290        POSIX_TRACE_NOT_TRUNCATED
14291        POSIX_TRACE_OVERFLOW
14292        POSIX_TRACE_OVERRUN
14293        POSIX_TRACE_RESUME
14294        POSIX_TRACE_RUNNING
14295        POSIX_TRACE_START
14296        POSIX_TRACE_STOP
14297        POSIX_TRACE_SUSPENDED
14298        POSIX_TRACE_SYSTEM_EVENTS
14299        POSIX_TRACE_TRUNCATED_READ

```

14300 POSIX\_TRACE\_TRUNCATED\_RECORD  
 14301 POSIX\_TRACE\_UNNAMED\_USER\_EVENT  
 14302 POSIX\_TRACE\_UNTIL\_FULL  
 14303 POSIX\_TRACE\_WOPID\_EVENTS

14304 OB TEF The <trace.h> header shall define the **size\_t**, **trace\_attr\_t**, **trace\_event\_id\_t**, **trace\_event\_set\_t**,  
 14305 and **trace\_id\_t** types as described in <sys/types.h>.

14306 The following shall be declared as functions and may also be defined as macros. Function  
 14307 prototypes shall be provided.

14308 int posix\_trace\_attr\_destroy(trace\_attr\_t \*);  
 14309 int posix\_trace\_attr\_getclockres(const trace\_attr\_t \*,  
 14310 struct timespec \*);  
 14311 int posix\_trace\_attr\_getcreatetime(const trace\_attr\_t \*,  
 14312 struct timespec \*);  
 14313 int posix\_trace\_attr\_getgenversion(const trace\_attr\_t \*, char \*);  
 14314 TRI int posix\_trace\_attr\_getinherited(const trace\_attr\_t \*restrict,  
 14315 int \*restrict);  
 14316 TRL int posix\_trace\_attr\_getlogfullpolicy(const trace\_attr\_t \*restrict,  
 14317 int \*restrict);  
 14318 int posix\_trace\_attr\_getlogsize(const trace\_attr\_t \*restrict,  
 14319 size\_t \*restrict);  
 14320 int posix\_trace\_attr\_getmaxdatasize(const trace\_attr\_t \*restrict,  
 14321 size\_t \*restrict);  
 14322 int posix\_trace\_attr\_getmaxsystemeventsize(const trace\_attr\_t \*restrict,  
 14323 size\_t \*restrict);  
 14324 int posix\_trace\_attr\_getmaxusereventsize(const trace\_attr\_t \*restrict,  
 14325 size\_t, size\_t \*restrict);  
 14326 int posix\_trace\_attr\_getname(const trace\_attr\_t \*, char \*);  
 14327 int posix\_trace\_attr\_getstreamfullpolicy(const trace\_attr\_t \*restrict,  
 14328 int \*restrict);  
 14329 int posix\_trace\_attr\_getstreamsize(const trace\_attr\_t \*restrict,  
 14330 size\_t \*restrict);  
 14331 int posix\_trace\_attr\_init(trace\_attr\_t \*);  
 14332 TRI int posix\_trace\_attr\_setinherited(trace\_attr\_t \*, int);  
 14333 TRL int posix\_trace\_attr\_setlogfullpolicy(trace\_attr\_t \*, int);  
 14334 int posix\_trace\_attr\_setlogsize(trace\_attr\_t \*, size\_t);  
 14335 int posix\_trace\_attr\_setmaxdatasize(trace\_attr\_t \*, size\_t);  
 14336 int posix\_trace\_attr\_setname(trace\_attr\_t \*, const char \*);  
 14337 int posix\_trace\_attr\_setstreamfullpolicy(trace\_attr\_t \*, int);  
 14338 int posix\_trace\_attr\_setstreamsize(trace\_attr\_t \*, size\_t);  
 14339 int posix\_trace\_clear(trace\_id\_t);  
 14340 TRL int posix\_trace\_close(trace\_id\_t);  
 14341 int posix\_trace\_create(pid\_t, const trace\_attr\_t \*restrict,  
 14342 trace\_id\_t \*restrict);  
 14343 TRL int posix\_trace\_create\_withlog(pid\_t, const trace\_attr\_t \*restrict,  
 14344 int, trace\_id\_t \*restrict);  
 14345 void posix\_trace\_event(trace\_event\_id\_t, const void \*restrict, size\_t);  
 14346 int posix\_trace\_eventid\_equal(trace\_id\_t, trace\_event\_id\_t,  
 14347 trace\_event\_id\_t);  
 14348 int posix\_trace\_eventid\_get\_name(trace\_id\_t, trace\_event\_id\_t, char \*);  
 14349 int posix\_trace\_eventid\_open(const char \*restrict,

## Headers

## &lt;trace.h&gt;

```

14350         trace_event_id_t *restrict);
14351 TEF int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14352     int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14353     int  posix_trace_eventset_empty(trace_event_set_t *);
14354     int  posix_trace_eventset_fill(trace_event_set_t *, int);
14355     int  posix_trace_eventset_ismember(trace_event_id_t,
14356         const trace_event_set_t *restrict, int *restrict);
14357     int  posix_trace_eventtypelist_getnext_id(trace_id_t,
14358         trace_event_id_t *restrict, int *restrict);
14359     int  posix_trace_eventtypelist_rewind(trace_id_t);
14360 TRL int  posix_trace_flush(trace_id_t);
14361     int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
14362 TEF int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14363     int  posix_trace_get_status(trace_id_t,
14364         struct posix_trace_status_info *);
14365     int  posix_trace_getnext_event(trace_id_t,
14366         struct posix_trace_event_info *restrict, void *restrict,
14367         size_t, size_t *restrict, int *restrict);
14368 TRL int  posix_trace_open(int, trace_id_t *);
14369     int  posix_trace_rewind(trace_id_t);
14370 TEF int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14371     int  posix_trace_shutdown(trace_id_t);
14372     int  posix_trace_start(trace_id_t);
14373     int  posix_trace_stop(trace_id_t);
14374     int  posix_trace_timedgetnext_event(trace_id_t,
14375         struct posix_trace_event_info *restrict, void *restrict,
14376         size_t, size_t *restrict, int *restrict,
14377         const struct timespec *restrict);
14378 TEF int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
14379         trace_event_id_t *restrict);
14380     int  posix_trace_trygetnext_event(trace_id_t,
14381         struct posix_trace_event_info *restrict, void *restrict, size_t,
14382         size_t *restrict, int *restrict);

```

## 14383 APPLICATION USAGE

14384 None.

## 14385 RATIONALE

14386 None.

## 14387 FUTURE DIRECTIONS

14388 The &lt;trace.h&gt; header may be removed in a future version.

## 14389 SEE ALSO

14390 &lt;sys/types.h&gt;

14391 XSH Section 2.11 (on page 529), *posix\_trace\_attr\_destroy()*, *posix\_trace\_attr\_getclockres()*,  
14392 *posix\_trace\_attr\_getinherited()*, *posix\_trace\_attr\_getlogsize()*, *posix\_trace\_clear()*, *posix\_trace\_close()*,  
14393 *posix\_trace\_create()*, *posix\_trace\_event()*, *posix\_trace\_eventid\_equal()*, *posix\_trace\_eventset\_add()*,  
14394 *posix\_trace\_eventtypelist\_getnext\_id()*, *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*,  
14395 *posix\_trace\_getnext\_event()*, *posix\_trace\_start()*

14396 **CHANGE HISTORY**

14397 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14398 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin  
14399 code to the *posix\_trace\_flush()* function, for alignment with the System Interfaces volume of  
14400 POSIX.1-2008.

14401 **Issue 7**

14402 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

14403 The <trace.h> header is marked obsolescent.

14404 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;ulimit.h&gt;

14405 **NAME**14406 `ulimit.h` — `ulimit` commands14407 **SYNOPSIS**14408 OB XSI `#include <ulimit.h>`14409 **DESCRIPTION**14410 The <**ulimit.h**> header shall define the symbolic constants used by the `ulimit()` function.

14411 Symbolic constants:

14412 `UL_GETFSIZE` Get maximum file size.14413 `UL_SETFSIZE` Set maximum file size.14414 The following shall be declared as a function and may also be defined as a macro. A function  
14415 prototype shall be provided.14416 `long ulimit(int, ...);`14417 **APPLICATION USAGE**14418 See `ulimit()`.14419 **RATIONALE**

14420 None.

14421 **FUTURE DIRECTIONS**

14422 None.

14423 **SEE ALSO**14424 XSH `ulimit()`14425 **CHANGE HISTORY**

14426 First released in Issue 3.

14427 **Issue 7**14428 The <**ulimit.h**> header is marked obsolescent.

**<unistd.h>**

Headers

14429 **NAME**

14430 unistd.h — standard symbolic constants and types

14431 **SYNOPSIS**

14432 #include &lt;unistd.h&gt;

14433 **DESCRIPTION**

14434 The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares  
 14435 miscellaneous functions. The actual values of the constants are unspecified except as shown. The  
 14436 contents of this header are shown below.

14437 **Version Test Macros**

14438 The **<unistd.h>** header shall define the following symbolic constants. The values shall be  
 14439 suitable for use in **#if** preprocessing directives.

14440 **\_POSIX\_VERSION**

14441 Integer value indicating version of this standard (C-language binding) to which the  
 14442 implementation conforms. For implementations conforming to POSIX.1-2008, the value  
 14443 shall be 200809L.

14444 **\_POSIX2\_VERSION**

14445 Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the  
 14446 implementation conforms. For implementations conforming to POSIX.1-2008, the value  
 14447 shall be 200809L.

14448 The **<unistd.h>** header shall define the following symbolic constant only if the implementation  
 14449 supports the XSI option; see [Section 2.1.4](#) (on page 19). If defined, its value shall be suitable for  
 14450 use in **#if** preprocessing directives.

14451 XSI **\_XOPEN\_VERSION**

14452 Integer value indicating version of the X/Open Portability Guide to which the  
 14453 implementation conforms. The value shall be 700.

14454 **Constants for Options and Option Groups**

14455 The following symbolic constants, if defined in **<unistd.h>**, shall have a value of  $-1$ ,  $0$ , or  
 14456 greater, unless otherwise specified below. The values shall be suitable for use in **#if**  
 14457 preprocessing directives.

14458 If a symbolic constant is not defined or is defined with the value  $-1$ , the option is not supported  
 14459 for compilation. If it is defined with a value greater than zero, the option shall always be  
 14460 supported when the application is executed. If it is defined with the value zero, the option shall  
 14461 be supported for compilation and might or might not be supported at runtime. See [Section 2.1.6](#)  
 14462 (on page 26) for further information about the conformance requirements of these three  
 14463 categories of support.

14464 ADV **\_POSIX\_ADVISORY\_INFO**

14465 The implementation supports the Advisory Information option. If this symbol is defined in  
 14466 **<unistd.h>**, it shall be defined to be  $-1$ ,  $0$ , or 200809L. The value of this symbol reported by  
 14467 *sysconf()* shall either be  $-1$  or 200809L.

14468 **\_POSIX\_ASYNCHRONOUS\_IO**

14469 The implementation supports asynchronous input and output. This symbol shall always be  
 14470 set to the value 200809L.

14471		<b>_POSIX_BARRIERS</b>
14472		The implementation supports barriers. This symbol shall always be set to the value
14473		200809L.
14474		<b>_POSIX_CHOWN_RESTRICTED</b>
14475		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and
14476		to changing the group ID of a file only to the effective group ID of the process or to one of
14477		its supplementary group IDs. This symbol shall be defined with a value other than -1.
14478		<b>_POSIX_CLOCK_SELECTION</b>
14479		The implementation supports clock selection. This symbol shall always be set to the value
14480		200809L.
14481	CPT	<b>_POSIX_CPUTIME</b>
14482		The implementation supports the Process CPU-Time Clocks option. If this symbol is
14483		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14484		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14485	FSC	<b>_POSIX_FSYNC</b>
14486		The implementation supports the File Synchronization option. If this symbol is defined in
14487		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14488		<i>sysconf()</i> shall either be -1 or 200809L.
14489		<b>_POSIX_IPV6</b>
14490		The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it
14491		shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall
14492		either be -1 or 200809L.
14493		<b>_POSIX_JOB_CONTROL</b>
14494		The implementation supports job control. This symbol shall always be set to a value greater
14495		than zero.
14496		<b>_POSIX_MAPPED_FILES</b>
14497		The implementation supports memory mapped Files. This symbol shall always be set to the
14498		value 200809L.
14499	ML	<b>_POSIX_MEMLOCK</b>
14500		The implementation supports the Process Memory Locking option. If this symbol is defined
14501		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14502		by <i>sysconf()</i> shall either be -1 or 200809L.
14503	MLR	<b>_POSIX_MEMLOCK_RANGE</b>
14504		The implementation supports the Range Memory Locking option. If this symbol is defined
14505		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14506		by <i>sysconf()</i> shall either be -1 or 200809L.
14507		<b>_POSIX_MEMORY_PROTECTION</b>
14508		The implementation supports memory protection. This symbol shall always be set to the
14509		value 200809L.
14510	MSG	<b>_POSIX_MESSAGE_PASSING</b>
14511		The implementation supports the Message Passing option. If this symbol is defined in
14512		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14513		<i>sysconf()</i> shall either be -1 or 200809L.
14514	MON	<b>_POSIX_MONOTONIC_CLOCK</b>
14515		The implementation supports the Monotonic Clock option. If this symbol is defined in
14516		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by

14517		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14518		<code>_POSIX_NO_TRUNC</code>
14519		Pathname components longer than <code>{NAME_MAX}</code> generate an error. This symbol shall be
14520		defined with a value other than <code>-1</code> .
14521	PIO	<code>_POSIX_PRIORITIZED_IO</code>
14522		The implementation supports the Prioritized Input and Output option. If this symbol is
14523		defined in <code>&lt;unistd.h&gt;</code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol
14524		reported by <code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14525	PS	<code>_POSIX_PRIORITY_SCHEDULING</code>
14526		The implementation supports the Process Scheduling option. If this symbol is defined in
14527		<code>&lt;unistd.h&gt;</code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14528		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14529	RS	<code>_POSIX_RAW_SOCKETS</code>
14530		The implementation supports the Raw Sockets option. If this symbol is defined in
14531		<code>&lt;unistd.h&gt;</code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14532		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14533		<code>_POSIX_READER_WRITER_LOCKS</code>
14534		The implementation supports read-write locks. This symbol shall always be set to the value
14535		<code>200809L</code> .
14536		<code>_POSIX_REALTIME_SIGNALS</code>
14537		The implementation supports realtime signals. This symbol shall always be set to the value
14538		<code>200809L</code> .
14539		<code>_POSIX_REGEX</code>
14540		The implementation supports the Regular Expression Handling option. This symbol shall
14541		always be set to a value greater than zero.
14542		<code>_POSIX_SAVED_IDS</code>
14543		Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be
14544		set to a value greater than zero.
14545		<code>_POSIX_SEMAPHORES</code>
14546		The implementation supports semaphores. This symbol shall always be set to the value
14547		<code>200809L</code> .
14548	SHM	<code>_POSIX_SHARED_MEMORY_OBJECTS</code>
14549		The implementation supports the Shared Memory Objects option. If this symbol is defined
14550		in <code>&lt;unistd.h&gt;</code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported
14551		by <code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14552		<code>_POSIX_SHELL</code>
14553		The implementation supports the POSIX shell. This symbol shall always be set to a value
14554		greater than zero.
14555	SPN	<code>_POSIX_SPAWN</code>
14556		The implementation supports the Spawn option. If this symbol is defined in <code>&lt;unistd.h&gt;</code> , it
14557		shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by <code>sysconf()</code> shall
14558		either be <code>-1</code> or <code>200809L</code> .
14559		<code>_POSIX_SPIN_LOCKS</code>
14560		The implementation supports spin locks. This symbol shall always be set to the value
14561		<code>200809L</code> .

## Headers

## &lt;unistd.h&gt;

14562	SS	<b>_POSIX_SPARADIC_SERVER</b>
14563		The implementation supports the Process Sporadic Server option. If this symbol is defined
14564		in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14565		by <i>sysconf()</i> shall either be -1 or 200809L.
14566	SIO	<b>_POSIX_SYNCHRONIZED_IO</b>
14567		The implementation supports the Synchronized Input and Output option. If this symbol is
14568		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14569		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14570	TSA	<b>_POSIX_THREAD_ATTR_STACKADDR</b>
14571		The implementation supports the Thread Stack Address Attribute option. If this symbol is
14572		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14573		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14574	TSS	<b>_POSIX_THREAD_ATTR_STACKSIZE</b>
14575		The implementation supports the Thread Stack Size Attribute option. If this symbol is
14576		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14577		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14578	TCT	<b>_POSIX_THREAD_CPU_TIME</b>
14579		The implementation supports the Thread CPU-Time Clocks option. If this symbol is
14580		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14581		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14582	TPI	<b>_POSIX_THREAD_PRIO_INHERIT</b>
14583		The implementation supports the Non-Robust Mutex Priority Inheritance option. If this
14584		symbol is defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this
14585		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14586	TPP	<b>_POSIX_THREAD_PRIO_PROTECT</b>
14587		The implementation supports the Non-Robust Mutex Priority Protection option. If this
14588		symbol is defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this
14589		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14590	TPS	<b>_POSIX_THREAD_PRIORITY_SCHEDULING</b>
14591		The implementation supports the Thread Execution Scheduling option. If this symbol is
14592		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14593		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14594	TSH	<b>_POSIX_THREAD_PROCESS_SHARED</b>
14595		The implementation supports the Thread Process-Shared Synchronization option. If this
14596		symbol is defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this
14597		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14598	RPI	<b>_POSIX_THREAD_ROBUST_PRIO_INHERIT</b>
14599		The implementation supports the Robust Mutex Priority Inheritance option. If this symbol
14600		is defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14601		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14602	RPP	<b>_POSIX_THREAD_ROBUST_PRIO_PROTECT</b>
14603		The implementation supports the Robust Mutex Priority Protection option. If this symbol is
14604		defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14605		reported by <i>sysconf()</i> shall either be -1 or 200809L.

14606		<code>_POSIX_THREAD_SAFE_FUNCTIONS</code>	The implementation supports thread-safe functions. This symbol shall always be set to the value 200809L.
14607			
14608			
14609	TSP	<code>_POSIX_THREAD_SPORADIC_SERVER</code>	The implementation supports the Thread Sporadic Server option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14610			
14611			
14612			
14613		<code>_POSIX_THREADS</code>	The implementation supports threads. This symbol shall always be set to the value 200809L.
14614			
14615			
14616		<code>_POSIX_TIMEOUTS</code>	The implementation supports timeouts. This symbol shall always be set to the value 200809L.
14617			
14618			
14619		<code>_POSIX_TIMERS</code>	The implementation supports timers. This symbol shall always be set to the value 200809L.
14620			
14621	OB TRC	<code>_POSIX_TRACE</code>	The implementation supports the Trace option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14622			
14623			
14624			
14625	OB TEF	<code>_POSIX_TRACE_EVENT_FILTER</code>	The implementation supports the Trace Event Filter option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14626			
14627			
14628			
14629	OB TRI	<code>_POSIX_TRACE_INHERIT</code>	The implementation supports the Trace Inherit option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14630			
14631			
14632			
14633	OB TRL	<code>_POSIX_TRACE_LOG</code>	The implementation supports the Trace Log option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14634			
14635			
14636			
14637	TYM	<code>_POSIX_TYPED_MEMORY_OBJECTS</code>	The implementation supports the Typed Memory Objects option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <code>sysconf()</code> shall either be -1 or 200809L.
14638			
14639			
14640			
14641	OB	<code>_POSIX_V6_ILP32_OFF32</code>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14642			
14643			
14644	OB	<code>_POSIX_V6_ILP32_OFFBIG</code>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.
14645			
14646			
14647	OB	<code>_POSIX_V6_LP64_OFF64</code>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> and 64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14648			
14649			

## Headers

## &lt;unistd.h&gt;

14650	OB	<b>_POSIX_V6_LPBIG_OFFBIG</b>	The implementation provides a C-language compilation environment with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14651			
14652			
14653		<b>_POSIX_V7_ILP32_OFF32</b>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14654			
14655			
14656		<b>_POSIX_V7_ILP32_OFFBIG</b>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.
14657			
14658			
14659		<b>_POSIX_V7_LP64_OFF64</b>	The implementation provides a C-language compilation environment with 32-bit <b>int</b> and 64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14660			
14661			
14662		<b>_POSIX_V7_LPBIG_OFFBIG</b>	The implementation provides a C-language compilation environment with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14663			
14664			
14665		<b>_POSIX2_C_BIND</b>	The implementation supports the C-Language Binding option. This symbol shall always have the value 200809L.
14666			
14667			
14668	CD	<b>_POSIX2_C_DEV</b>	The implementation supports the C-Language Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14669			
14670			
14671			
14672		<b>_POSIX2_CHAR_TERM</b>	The implementation supports the Terminal Characteristics option. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or a value greater than zero.
14673			
14674			
14675	FD	<b>_POSIX2_FORT_DEV</b>	The implementation supports the FORTRAN Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14676			
14677			
14678			
14679	FR	<b>_POSIX2_FORT_RUN</b>	The implementation supports the FORTRAN Runtime Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14680			
14681			
14682			
14683		<b>_POSIX2_LOCALEDEF</b>	The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14684			
14685			
14686			
14687	OB BE	<b>_POSIX2_PBS</b>	The implementation supports the Batch Environment Services and Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14688			
14689			
14690			
14691	OB BE	<b>_POSIX2_PBS_ACCOUNTING</b>	The implementation supports the Batch Accounting option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14692			
14693			
14694			

## &lt;unistd.h&gt;

## Headers

14695	OB BE	<u>_POSIX2_PBS_CHECKPOINT</u>
14696		The implementation supports the Batch Checkpoint/Restart option. If this symbol is
14697		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14698		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14699	OB BE	<u>_POSIX2_PBS_LOCATE</u>
14700		The implementation supports the Locate Batch Job Request option. If this symbol is defined
14701		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14702		by <i>sysconf</i> () shall either be -1 or 200809L.
14703	OB BE	<u>_POSIX2_PBS_MESSAGE</u>
14704		The implementation supports the Batch Job Message Request option. If this symbol is
14705		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14706		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14707	OB BE	<u>_POSIX2_PBS_TRACK</u>
14708		The implementation supports the Track Batch Job Request option. If this symbol is defined
14709		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14710		by <i>sysconf</i> () shall either be -1 or 200809L.
14711	SD	<u>_POSIX2_SW_DEV</u>
14712		The implementation supports the Software Development Utilities option. If this symbol is
14713		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14714		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14715	UP	<u>_POSIX2_UPE</u>
14716		The implementation supports the User Portability Utilities option. If this symbol is defined
14717		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14718		by <i>sysconf</i> () shall either be -1 or 200809L.
14719	XSI	<u>_XOPEN_CRYPT</u>
14720		The implementation supports the X/Open Encryption Option Group.
14721		<u>_XOPEN_ENH_I18N</u>
14722		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14723		Group. This symbol shall always be set to a value other than -1.
14724		<u>_XOPEN_REALTIME</u>
14725		The implementation supports the X/Open Realtime Option Group.
14726		<u>_XOPEN_REALTIME_THREADS</u>
14727		The implementation supports the X/Open Realtime Threads Option Group.
14728		<u>_XOPEN_SHM</u>
14729		The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14730		symbol shall always be set to a value other than -1.
14731	OB XSR	<u>_XOPEN_STREAMS</u>
14732		The implementation supports the XSI STREAMS Option Group.
14733	XSI	<u>_XOPEN_UNIX</u>
14734		The implementation supports the XSI option.
14735	UU	<u>_XOPEN_UUCP</u>
14736		The implementation supports the UUCP Utilities option. If this symbol is defined in
14737		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14738		<i>sysconf</i> () shall be either -1 or 200809L.

14739 **Execution-Time Symbolic Constants**

14740 If any of the following symbolic constants are not defined in the <unistd.h> header, the value  
 14741 shall vary depending on the file to which it is applied. If defined, they shall have values suitable  
 14742 for use in #if preprocessing directives.

14743 If any of the following symbolic constants are defined to have value -1 in the <unistd.h> header,  
 14744 the implementation shall not provide the option on any file; if any are defined to have a value  
 14745 other than -1 in the <unistd.h> header, the implementation shall provide the option on all  
 14746 applicable files.

14747 All of the following values, whether defined as symbolic constants in <unistd.h> or not, may be  
 14748 queried with respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14749 `_POSIX_ASYNC_IO`

14750 Asynchronous input or output operations may be performed for the associated file.

14751 `_POSIX_PRIO_IO`

14752 Prioritized input or output operations may be performed for the associated file.

14753 `_POSIX_SYNC_IO`

14754 Synchronized input or output operations may be performed for the associated file.

14755 If the following symbolic constants are defined in the <unistd.h> header, they apply to files and  
 14756 all paths in all file systems on the implementation:

14757 `_POSIX_TIMESTAMP_RESOLUTION`

14758 The resolution in nanoseconds for all file timestamps.

14759 `_POSIX2_SYMLINKS`

14760 Symbolic links can be created.

14761 **Constants for Functions**

14762 The <unistd.h> header shall define NULL as described in <stddef.h>.

14763 The <unistd.h> header shall define the following symbolic constants for use with the *access()*  
 14764 function. The values shall be suitable for use in #if preprocessing directives.

14765 `F_OK` Test for existence of file.

14766 `R_OK` Test for read permission.

14767 `W_OK` Test for write permission.

14768 `X_OK` Test for execute (search) permission.

14769 The constants `F_OK`, `R_OK`, `W_OK`, and `X_OK` and the expressions `R_OK | W_OK`, `R_OK | X_OK`,  
 14770 and `R_OK | W_OK | X_OK` shall all have distinct values.

14771 The <unistd.h> header shall define the following symbolic constants for the *confstr()* function:

14772 `_CS_PATH`

14773 This is the value for the *PATH* environment variable that finds all standard utilities.

14774 `_CS_POSIX_V7_ILP32_OFF32_CFLAGS`

14775 If *sysconf(\_SC\_V7\_ILP32\_OFF32)* returns -1, the meaning of this value is unspecified.  
 14776 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an  
 14777 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

- 14778 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`  
 14779 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.  
 14780 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14781 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.
- 14782 `_CS_POSIX_V7_ILP32_OFF32_LIBS`  
 14783 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.  
 14784 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14785 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.
- 14786 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`  
 14787 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14788 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an  
 14789 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14790 **off\_t** type using at least 64 bits.
- 14791 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`  
 14792 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14793 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14794 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14795 **off\_t** type using at least 64 bits.
- 14796 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`  
 14797 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14798 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14799 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14800 **off\_t** type using at least 64 bits.
- 14801 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`  
 14802 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14803 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an  
 14804 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t**  
 14805 types.
- 14806 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`  
 14807 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14808 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14809 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t**  
 14810 types.
- 14811 `_CS_POSIX_V7_LP64_OFF64_LIBS`  
 14812 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14813 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14814 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t**  
 14815 types.
- 14816 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`  
 14817 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14818 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an  
 14819 application using a programming model with an **int** type using at least 32 bits and **long**,  
 14820 **pointer**, and **off\_t** types using at least 64 bits.
- 14821 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`  
 14822 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14823 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14824 application using a programming model with an **int** type using at least 32 bits and **long**,

- 14825 **pointer**, and **off\_t** types using at least 64 bits.
- 14826 `_CS_POSIX_V7_LPBIG_OFFBIG_LIBS`  
 14827 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14828 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14829 application using a programming model with an **int** type using at least 32 bits and **long**,  
 14830 **pointer**, and **off\_t** types using at least 64 bits.
- 14831 `_CS_POSIX_V7_THREADS_CFLAGS`  
 14832 If `sysconf(_SC_POSIX_THREADS)` returns `-1`, the meaning of this value is unspecified.  
 14833 Otherwise, this value is the set of initial options to be given to the `c99` utility to build a  
 14834 multi-threaded application. These flags are in addition to those associated with any of the  
 14835 other `_CS_POSIX_V7*_CFLAGS` values used to specify particular type size programming  
 14836 environments.
- 14837 `_CS_POSIX_V7_THREADS_LDFLAGS`  
 14838 If `sysconf(_SC_POSIX_THREADS)` returns `-1`, the meaning of this value is unspecified.  
 14839 Otherwise, this value is the set of final options to be given to the `c99` utility to build a multi-  
 14840 threaded application. These flags are in addition to those associated with any of the other  
 14841 `_CS_POSIX_V7*_LDFLAGS` values used to specify particular type size programming  
 14842 environments.
- 14843 `_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS`  
 14844 This value is a <newline>-separated list of names of programming environments supported  
 14845 by the implementation in which the widths of the **blksize\_t**, **cc\_t**, **mode\_t**, **nfds\_t**, **pid\_t**,  
 14846 **ptrdiff\_t**, **size\_t**, **speed\_t**, **ssize\_t**, **suseconds\_t**, **tcflag\_t**, **wchar\_t**, and **wint\_t** types are no  
 14847 greater than the width of type **long**. The format of each name shall be suitable for use with  
 14848 the `getconf -v` option.
- 14849 `_CS_V7_ENV`  
 14850 This is the value that provides the environment variable information (other than that  
 14851 provided by `_CS_PATH`) that is required by the implementation to create a conforming  
 14852 environment, as described in the implementation's conformance documentation.
- 14853 OB The following symbolic constants are reserved for compatibility with Issue 6:
- 14854 `_CS_POSIX_V6_ILP32_OFF32_CFLAGS`  
 14855 `_CS_POSIX_V6_ILP32_OFF32_LDFLAGS`  
 14856 `_CS_POSIX_V6_ILP32_OFF32_LIBS`  
 14857 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS`  
 14858 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS`  
 14859 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS`  
 14860 `_CS_POSIX_V6_LP64_OFF64_CFLAGS`  
 14861 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`  
 14862 `_CS_POSIX_V6_LP64_OFF64_LIBS`  
 14863 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS`  
 14864 `_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS`  
 14865 `_CS_POSIX_V6_LPBIG_OFFBIG_LIBS`  
 14866 `_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS`  
 14867 `_CS_V6_ENV`
- 14868 The <unistd.h> header shall define `SEEK_CUR`, `SEEK_END`, and `SEEK_SET` as described in  
 14869 <stdio.h>.
- 14870 The <unistd.h> header shall define the following symbolic constants as possible values for the

## &lt;unistd.h&gt;

14871	<i>function</i> argument to the <i>lockf()</i> function:
14872	F_LOCK            Lock a section for exclusive use.
14873	F_TEST            Test section for locks by other processes.
14874	F_TLOCK          Test and lock a section for exclusive use.
14875	F_ULOCK          Unlock locked sections.
14876	The <unistd.h> header shall define the following symbolic constants for <i>pathconf()</i> :
14877	_PC_2_SYMLINKS
14878	_PC_ALLOC_SIZE_MIN
14879	_PC_ASYNC_IO
14880	_PC_CHOWN_RESTRICTED
14881	_PC_FILESIZEBITS
14882	_PC_LINK_MAX
14883	_PC_MAX_CANON
14884	_PC_MAX_INPUT
14885	_PC_NAME_MAX
14886	_PC_NO_TRUNC
14887	_PC_PATH_MAX
14888	_PC_PIPE_BUF
14889	_PC_PRIO_IO
14890	_PC_REC_INCR_XFER_SIZE
14891	_PC_REC_MAX_XFER_SIZE
14892	_PC_REC_MIN_XFER_SIZE
14893	_PC_REC_XFER_ALIGN
14894	_PC_SYMLINK_MAX
14895	_PC_SYNC_IO
14896	_PC_TIMESTAMP_RESOLUTION
14897	_PC_VDISABLE
14898	The <unistd.h> header shall define the following symbolic constants for <i>sysconf()</i> :
14899	_SC_2_C_BIND
14900	_SC_2_C_DEV
14901	_SC_2_CHAR_TERM
14902	_SC_2_FORT_DEV
14903	_SC_2_FORT_RUN
14904	_SC_2_LOCALEDEF
14905	_SC_2_PBS
14906	_SC_2_PBS_ACCOUNTING
14907	_SC_2_PBS_CHECKPOINT
14908	_SC_2_PBS_LOCATE
14909	_SC_2_PBS_MESSAGE
14910	_SC_2_PBS_TRACK
14911	_SC_2_SW_DEV
14912	_SC_2_UPE
14913	_SC_2_VERSION
14914	_SC_ADVISORY_INFO
14915	_SC_AIO_LISTIO_MAX
14916	_SC_AIO_MAX
14917	_SC_AIO_PRIO_DELTA_MAX
14918	_SC_ARG_MAX

## Headers

&lt;unistd.h&gt;

14919 \_SC\_ASYNCHRONOUS\_IO  
 14920 \_SC\_ATEXIT\_MAX  
 14921 \_SC\_BARRIERS  
 14922 \_SC\_BC\_BASE\_MAX  
 14923 \_SC\_BC\_DIM\_MAX  
 14924 \_SC\_BC\_SCALE\_MAX  
 14925 \_SC\_BC\_STRING\_MAX  
 14926 \_SC\_CHILD\_MAX  
 14927 \_SC\_CLK\_TCK  
 14928 \_SC\_CLOCK\_SELECTION  
 14929 \_SC\_COLL\_WEIGHTS\_MAX  
 14930 \_SC\_CPUTIME  
 14931 \_SC\_DELAYTIMER\_MAX  
 14932 \_SC\_EXPR\_NEST\_MAX  
 14933 \_SC\_FSYNC  
 14934 \_SC\_GETGR\_R\_SIZE\_MAX  
 14935 \_SC\_GETPW\_R\_SIZE\_MAX  
 14936 \_SC\_HOST\_NAME\_MAX  
 14937 \_SC\_IOV\_MAX  
 14938 \_SC\_IPV6  
 14939 \_SC\_JOB\_CONTROL  
 14940 \_SC\_LINE\_MAX  
 14941 \_SC\_LOGIN\_NAME\_MAX  
 14942 \_SC\_MAPPED\_FILES  
 14943 \_SC\_MEMLOCK  
 14944 \_SC\_MEMLOCK\_RANGE  
 14945 \_SC\_MEMORY\_PROTECTION  
 14946 \_SC\_MESSAGE\_PASSING  
 14947 \_SC\_MONOTONIC\_CLOCK  
 14948 \_SC\_MQ\_OPEN\_MAX  
 14949 \_SC\_MQ\_PRIO\_MAX  
 14950 \_SC\_NGROUPS\_MAX  
 14951 \_SC\_OPEN\_MAX  
 14952 \_SC\_PAGE\_SIZE  
 14953 \_SC\_PAGESIZE  
 14954 \_SC\_PRIORITIZED\_IO  
 14955 \_SC\_PRIORITY\_SCHEDULING  
 14956 \_SC\_RAW\_SOCKETS  
 14957 \_SC\_READ\_DUP\_MAX  
 14958 \_SC\_READER\_WRITER\_LOCKS  
 14959 \_SC\_REALTIME\_SIGNALS  
 14960 \_SC\_REGEX  
 14961 \_SC\_RT\_SIG\_MAX  
 14962 \_SC\_SAVED\_IDS  
 14963 \_SC\_SEM\_NSEMS\_MAX  
 14964 \_SC\_SEM\_VALUE\_MAX  
 14965 \_SC\_SEMAPHORES  
 14966 \_SC\_SHARED\_MEMORY\_OBJECTS  
 14967 \_SC\_SHELL  
 14968 \_SC\_SIGQUEUE\_MAX  
 14969 \_SC\_SPAWN  
 14970 \_SC\_SPIN\_LOCKS

14971 \_SC\_SPORADIC\_SERVER  
 14972 \_SC\_SS\_REPL\_MAX  
 14973 \_SC\_STREAM\_MAX  
 14974 \_SC\_SYMLOOP\_MAX  
 14975 \_SC\_SYNCHRONIZED\_IO  
 14976 \_SC\_THREAD\_ATTR\_STACKADDR  
 14977 \_SC\_THREAD\_ATTR\_STACKSIZE  
 14978 \_SC\_THREAD\_CPU\_TIME  
 14979 \_SC\_THREAD\_DESTRUCTOR\_ITERATIONS  
 14980 \_SC\_THREAD\_KEYS\_MAX  
 14981 \_SC\_THREAD\_PRIO\_INHERIT  
 14982 \_SC\_THREAD\_PRIO\_PROTECT  
 14983 \_SC\_THREAD\_PRIORITY\_SCHEDULING  
 14984 \_SC\_THREAD\_PROCESS\_SHARED  
 14985 \_SC\_THREAD\_ROBUST\_PRIO\_INHERIT  
 14986 \_SC\_THREAD\_ROBUST\_PRIO\_PROTECT  
 14987 \_SC\_THREAD\_SAFE\_FUNCTIONS  
 14988 \_SC\_THREAD\_SPORADIC\_SERVER  
 14989 \_SC\_THREAD\_STACK\_MIN  
 14990 \_SC\_THREAD\_THREADS\_MAX  
 14991 \_SC\_THREADS  
 14992 \_SC\_TIMEOUTS  
 14993 \_SC\_TIMER\_MAX  
 14994 \_SC\_TIMERS  
 14995 \_SC\_TRACE  
 14996 \_SC\_TRACE\_EVENT\_FILTER  
 14997 \_SC\_TRACE\_EVENT\_NAME\_MAX  
 14998 \_SC\_TRACE\_INHERIT  
 14999 \_SC\_TRACE\_LOG  
 15000 \_SC\_TRACE\_NAME\_MAX  
 15001 \_SC\_TRACE\_SYS\_MAX  
 15002 \_SC\_TRACE\_USER\_EVENT\_MAX  
 15003 \_SC\_TTY\_NAME\_MAX  
 15004 \_SC\_TYPED\_MEMORY\_OBJECTS  
 15005 \_SC\_TZNAME\_MAX  
 15006 \_SC\_V7\_ILP32\_OFF32  
 15007 \_SC\_V7\_ILP32\_OFFBIG  
 15008 \_SC\_V7\_LP64\_OFF64  
 15009 \_SC\_V7\_LPBIG\_OFFBIG  
 15010 OB \_SC\_V6\_ILP32\_OFF32  
 15011 \_SC\_V6\_ILP32\_OFFBIG  
 15012 \_SC\_V6\_LP64\_OFF64  
 15013 \_SC\_V6\_LPBIG\_OFFBIG  
 15014 \_SC\_VERSION  
 15015 \_SC\_XOPEN\_CRYPT  
 15016 \_SC\_XOPEN\_ENH\_I18N  
 15017 \_SC\_XOPEN\_REALTIME  
 15018 \_SC\_XOPEN\_REALTIME\_THREADS  
 15019 \_SC\_XOPEN\_SHM  
 15020 \_SC\_XOPEN\_STREAMS  
 15021 \_SC\_XOPEN\_UNIX  
 15022 \_SC\_XOPEN\_UUCP

15023 `_SC_XOPEN_VERSION`

15024 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same value.

15025 The <unistd.h> header shall define the following symbolic constants for file streams:

15026 `STDERR_FILENO` File number of *stderr*; 2.

15027 `STDIN_FILENO` File number of *stdin*; 0.

15028 `STDOUT_FILENO` File number of *stdout*; 1.

15029 The <unistd.h> header shall define the following symbolic constant for terminal special  
15030 character handling:

15031 `_POSIX_VDISABLE` This symbol shall be defined to be the value of a character that shall  
15032 disable terminal special character handling as described in Section 11.2.6  
15033 (on page 212). This symbol shall always be set to a value other than -1.

### 15034 Type Definitions

15035 The <unistd.h> header shall define the `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types as  
15036 described in <sys/types.h>.

15037 The <unistd.h> header shall define the `intptr_t` type as described in <inttypes.h>.

### 15038 Declarations

15039 The following shall be declared as functions and may also be defined as macros. Function  
15040 prototypes shall be provided.

```

15041 int      access(const char *, int);
15042 unsigned alarm(unsigned);
15043 int      chdir(const char *);
15044 int      chown(const char *, uid_t, gid_t);
15045 int      close(int);
15046 size_t   confstr(int, char *, size_t);
15047 XSI     char *crypt(const char *, const char *);
15048 CX      char *ctermid(char *);
15049 int      dup(int);
15050 int      dup2(int, int);
15051 void     _exit(int);
15052 XSI     void encrypt(char [64], int);
15053 int      execl(const char *, const char *, ...);
15054 int      execlp(const char *, const char *, ...);
15055 int      execlp(const char *, const char *, ...);
15056 int      execv(const char *, char *const []);
15057 int      execve(const char *, char *const [], char *const []);
15058 int      execvp(const char *, char *const []);
15059 int      faccessat(int, const char *, int, int);
15060 int      fchdir(int);
15061 int      fchown(int, uid_t, gid_t);
15062 int      fchownat(int, const char *, uid_t, gid_t, int);
15063 SIO     int fdatsync(int);
15064 int      fexecve(int, char *const [], char *const []);
15065 pid_t    fork(void);
15066 long     fpathconf(int, int);

```

15067	FSC	int	fsync(int);
15068		int	ftruncate(int, off_t);
15069		char	*getcwd(char *, size_t);
15070		gid_t	getegid(void);
15071		uid_t	geteuid(void);
15072		gid_t	getgid(void);
15073		int	getgroups(int, gid_t []);
15074	XSI	long	gethostid(void);
15075		int	gethostname(char *, size_t);
15076		char	*getlogin(void);
15077		int	getlogin_r(char *, size_t);
15078		int	getopt(int, char * const [], const char *);
15079		pid_t	getpgid(pid_t);
15080		pid_t	getpgrp(void);
15081		pid_t	getpid(void);
15082		pid_t	getppid(void);
15083		pid_t	getsid(pid_t);
15084		uid_t	getuid(void);
15085		int	isatty(int);
15086		int	lchown(const char *, uid_t, gid_t);
15087		int	link(const char *, const char *);
15088		int	linkat(int, const char *, int, const char *, int);
15089	XSI	int	lockf(int, int, off_t);
15090		off_t	lseek(int, off_t, int);
15091	XSI	int	nice(int);
15092		long	pathconf(const char *, int);
15093		int	pause(void);
15094		int	pipe(int [2]);
15095		ssize_t	pread(int, void *, size_t, off_t);
15096		ssize_t	pwrite(int, const void *, size_t, off_t);
15097		ssize_t	read(int, void *, size_t);
15098		ssize_t	readlink(const char *restrict, char *restrict, size_t);
15099		ssize_t	readlinkat(int, const char *restrict, char *restrict, size_t);
15100		int	rmdir(const char *);
15101		int	setegid(gid_t);
15102		int	seteuid(uid_t);
15103		int	setgid(gid_t);
15104		int	setpgid(pid_t, pid_t);
15105	OB XSI	pid_t	setpgrp(void);
15106	XSI	int	setregid(gid_t, gid_t);
15107		int	setreuid(uid_t, uid_t);
15108		pid_t	setsid(void);
15109		int	setuid(uid_t);
15110		unsigned	sleep(unsigned);
15111	XSI	void	swab(const void *restrict, void *restrict, ssize_t);
15112		int	symlink(const char *, const char *);
15113		int	symlinkat(const char *, int, const char *);
15114	XSI	void	sync(void);
15115		long	sysconf(int);
15116		pid_t	tcgetpgrp(int);
15117		int	tcsetpgrp(int, pid_t);
15118		int	truncate(const char *, off_t);

```

15119     char          *ttyname(int);
15120     int           ttyname_r(int, char *, size_t);
15121     int           unlink(const char *);
15122     int           unlinkat(int, const char *, int);
15123     ssize_t       write(int, const void *, size_t);

```

15124 Implementations may also include the *pthread\_atfork()* prototype as defined in <pthread.h>.

15125 The <unistd.h> header shall declare the following external variables:

```

15126     extern char   *optarg;
15127     extern int    opterr, optind, optopt;

```

#### 15128 APPLICATION USAGE

15129 POSIX.1-2008 only describes the behavior of systems that claim conformance to it. However,  
 15130 application developers who want to write applications that adapt to other versions of this  
 15131 standard (or to systems that do not conform to any POSIX standard) may find it useful to code  
 15132 them so as to conditionally compile different code depending on the value of  
 15133 `_POSIX_VERSION`, for example:

```

15134     #if _POSIX_VERSION >= 200112L
15135     /* Use the newer function that copes with large files. */
15136     off_t pos=ftello(fp);
15137     #else
15138     /* Either this is an old version of POSIX, or _POSIX_VERSION is
15139        not even defined, so use the traditional function. */
15140     long pos=ftell(fp);
15141     #endif

```

15142 Earlier versions of POSIX.1-2008 and of the Single UNIX Specification can be identified by the  
 15143 following macros:

```

15144     POSIX.1-1988 standard
15145     _POSIX_VERSION == 198808L

```

```

15146     POSIX.1-1990 standard
15147     _POSIX_VERSION == 199009L

```

```

15148     ISO POSIX-1: 1996 standard
15149     _POSIX_VERSION == 199506L

```

```

15150     Single UNIX Specification, Version 1
15151     _XOPEN_UNIX and _XOPEN_VERSION == 4

```

```

15152     Single UNIX Specification, Version 2
15153     _XOPEN_UNIX and _XOPEN_VERSION == 500

```

```

15154     ISO POSIX-1: 2001 and Single UNIX Specification, Version 3
15155     _POSIX_VERSION == 200112L, plus (if the XSI option is supported) _XOPEN_UNIX and
15156     _XOPEN_VERSION == 600

```

15157 POSIX.1-2008 does not make any attempt to define application binary interaction with the  
 15158 underlying operating system. However, application developers may find it useful to query  
 15159 `_SC_VERSION` at runtime via *sysconf()* to determine whether the current version of the  
 15160 operating system supports the necessary functionality as in the following program fragment:

```

15161     if (sysconf(_SC_VERSION) < 200809L) {
15162         fprintf(stderr, "POSIX.1-2008 system required, terminating \n");
15163         exit(1);

```

15164            }

15165            New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

#### 15166 **RATIONALE**

15167            As POSIX.1-2008 evolved, certain options became sufficiently standardized that it was  
15168            concluded that simply requiring one of the option choices was simpler than retaining the option.  
15169            However, for backwards-compatibility, the option flags (with required constant values) are  
15170            retained.

#### 15171 **Version Test Macros**

15172            The standard developers considered altering the definition of `_POSIX_VERSION` and removing  
15173            `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed  
15174            by some to be minimal, and since the implementation of the functionality is potentially  
15175            problematic. However, they recognized that support for existing application binaries is a  
15176            concern to manufacturers, application developers, and the users of implementations conforming  
15177            to POSIX.1-2008.

15178            While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide  
15179            the greatest degree of imaginable utility to the application developer or user, it is arguably better  
15180            than a **core** file or some other equally obscure result. (It is also possible for implementations to  
15181            encode and recognize application binaries compiled in various POSIX.1-conforming  
15182            environments, and modify the semantics of the underlying system to conform to the  
15183            expectations of the application.) For the reasons outlined in the preceding paragraphs and in the  
15184            APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`  
15185            and `_SC_VERSION` functionality.

#### 15186 **Compile-Time Symbolic Constants for System-Wide Options**

15187            POSIX.1-2008 includes support in certain areas for the newly adopted policy governing options  
15188            and stubs.

15189            This policy provides flexibility for implementations in how they support options. It also  
15190            specifies how conforming applications can adapt to different implementations that support  
15191            different sets of options. It allows the following:

- 15192            1. If an implementation has no interest in supporting an option, it does not have to provide  
15193            anything associated with that option beyond the announcement that it does not support  
15194            it.
- 15195            2. An implementation can support a partial or incompatible version of an option (as a non-  
15196            standard extension) as long as it does not claim to support the option.
- 15197            3. An application can determine whether the option is supported. A strictly conforming  
15198            application must check this announcement mechanism before first using anything  
15199            associated with the option.

15200            There is an important implication of this policy. POSIX.1-2008 cannot dictate the behavior of  
15201            interfaces associated with an option when the implementation does not claim to support the  
15202            option. In particular, it cannot require that a function associated with an unsupported option  
15203            will fail if it does not perform as specified. However, this policy does not prevent a standard  
15204            from requiring certain functions to always be present, but that they shall always fail on some  
15205            implementations. The `setpgid()` function in the POSIX.1-1990 standard, for example, is  
15206            considered appropriate.

15207            The POSIX standards include various options, and the C-language binding support for an

15208 option implies that the implementation must supply data types and function interfaces. An  
 15209 application must be able to discover whether the implementation supports each option.

15210 Any application must consider the following three cases for each option:

15211 1. Option never supported.

15212 The implementation advertises at compile time that the option will never be supported.  
 15213 In this case, it is not necessary for the implementation to supply any of the data types or  
 15214 function interfaces that are provided only as part of the option. The implementation  
 15215 might provide data types and functions that are similar to those defined by POSIX.1-2008,  
 15216 but there is no guarantee for any particular behavior.

15217 2. Option always supported.

15218 The implementation advertises at compile time that the option will always be supported.  
 15219 In this case, all data types and function interfaces shall be available and shall operate as  
 15220 specified.

15221 3. Option might or might not be supported.

15222 Some implementations might not provide a mechanism to specify support of options at  
 15223 compile time. In addition, the implementation might be unable or unwilling to specify  
 15224 support or non-support at compile time. In either case, any application that might use the  
 15225 option at runtime must be able to compile and execute. The implementation must  
 15226 provide, at compile time, all data types and function interfaces that are necessary to allow  
 15227 this. In this situation, there must be a mechanism that allows the application to query, at  
 15228 runtime, whether the option is supported. If the application attempts to use the option  
 15229 when it is not supported, the result is unspecified unless explicitly specified otherwise in  
 15230 POSIX.1-2008.

#### 15231 FUTURE DIRECTIONS

15232 None.

#### 15233 SEE ALSO

15234 <inttypes.h>, <limits.h>, <stddef.h>, <stdio.h>, <sys/socket.h>, <sys/types.h>, <termios.h>,  
 15235 <wctype.h>

15236 XSH *access()*, *alarm()*, *chown()*, *close()*, *confstr()*, *crypt()*, *ctermid()*, *dup()*, *\_Exit()*, *encrypt()*, *exec*,  
 15237 *fchdir()*, *fchown()*, *fdatasync()*, *fork()*, *fpathconf()*, *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*,  
 15238 *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*, *getlogin()*, *getopt()*, *getpgid()*, *getpgrp()*,  
 15239 *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*, *lockf()*, *lseek()*, *nice()*, *pause()*,  
 15240 *pipe()*, *read()*, *readlink()*, *rmdir()*, *setegid()*, *seteuid()*, *setgid()*, *setpgid()*, *setpgrp()*, *setregid()*,  
 15241 *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*,  
 15242 *truncate()*, *ttyname()*, *unlink()*, *write()*

#### 15243 CHANGE HISTORY

15244 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 15245 Issue 5

15246 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 15247 Threads Extension.

15248 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.  
 15249 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other  
 15250 than `-1` by a conforming implementation.

15251 Large File System extensions are added.

- 15252 The type of the argument to *sbrk()* is changed from **int** to **intptr\_t**.
- 15253 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of  
15254 constants for the *confstr()* function, and to the list of constants to the *sysconf()* function. These  
15255 are all marked EX.
- 15256 **Issue 6**
- 15257 `_POSIX2_C_VERSION` is removed.
- 15258 The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync()*.
- 15259 The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`,  
15260 `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.
- 15261 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI  
15262 STREAMS Option Group.
- 15263 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in  
15264 IEEE Std 1003.1-2001.
- 15265 The LEGACY symbol `_SC_PASS_MAX` is removed.
- 15266 The following new requirements on POSIX implementations derive from alignment with the  
15267 Single UNIX Specification:
- 15268 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the *confstr()* function.
  - 15269 • The `_SC_XBS5_*` constants are added for the *sysconf()* function.
  - 15270 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
  - 15271 • The **uid\_t**, **gid\_t**, **off\_t**, **pid\_t**, and **useconds\_t** types are mandated.
- 15272 The *gethostname()* prototype is added for sockets.
- 15273 A new section is added for System-Wide Options.
- 15274 Function prototypes for *setegid()* and *seteuid()* are added.
- 15275 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,  
15276 `_POSIX_SPAWN`, `_POSIX_SPARADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,  
15277 `_POSIX_THREAD_SPARADIC_SERVER`, and `_POSIX_TIMEOUTS`, and *pathconf()* variables are  
15278 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,  
15279 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with IEEE Std  
15280 1003.1d-1999.
- 15281 The following are added for alignment with IEEE Std 1003.1j-2000:
- 15282 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,  
15283 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,  
15284 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
  - 15285 • *sysconf()* variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,  
15286 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and  
15287 `_SC_TYPED_MEMORY_OBJECTS`
- 15288 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,  
15289 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are  
15290 introduced.
- 15291 The *getwd()* function is marked LEGACY.
- 15292 The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*.

- 15293 Constants for options are now harmonized, so when supported they take the year of approval of  
15294 IEEE Std 1003.1-2001 as the value.
- 15295 The following are added for alignment with IEEE Std 1003.1q-2000:
- 15296 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,  
15297 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
  - 15298 • The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,  
15299 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`
- 15300 The `brk()` and `sbrk()` LEGACY functions are removed.
- 15301 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning  
15302 information.
- 15303 The Open Group Base Resolution bwg2001-008 is applied, changing the `namelen` parameter for  
15304 `gethostname()` from `socklen_t` to `size_t`.
- 15305 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack  
15306 Address Size” to “Thread Stack Size Attribute”.
- 15307 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`,  
15308 `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.
- 15309 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description  
15310 in “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`,  
15311 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.
- 15312 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for  
15313 the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.
- 15314 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the  
15315 `_PC_SYMLINK_MAX` and `_SC_SYMLINK_MAX` constants.
- 15316 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and  
15317 margin code for the `fsync()` function.
- 15318 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to  
15319 the APPLICATION USAGE section: “New applications should not use `_XOPEN_SHM` or  
15320 `_XOPEN_ENH_I18N`.”.
- 15321 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements  
15322 for when constants for Options and Option Groups can be defined or undefined.
- 15323 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the  
15324 `_V6_ILP32_OFF32`, `_V6_ILP32_OFFBIG`, `_V6_LP64_OFF64`, and `_V6_LP64_OFFBIG` symbols to  
15325 `_POSIX_V6_ILP32_OFF32`, `_POSIX_V6_ILP32_OFFBIG`, `_POSIX_V6_LP64_OFF64`, and  
15326 `_POSIX_V6_LP64_OFFBIG`, respectively. This is for consistency with the `sysconf()` and `c99`  
15327 reference pages.
- 15328 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of  
15329 names of programming environments can be obtained using the `getconf -v` option.
- 15330 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the  
15331 `_SC_FILE_LOCKING`, `_SC_2_C_VERSION`, and `_SC_XOPEN_XCU_VERSION` constants.
- 15332 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding  
15333 `_SC_SS_REPL_MAX`, `_SC_TRACE_EVENT_NAME_MAX`, `_SC_TRACE_NAME_MAX`,  
15334 `_SC_TRACE_SYS_MAX`, and `_SC_TRACE_USER_EVENT_MAX` to the list of symbolic constants  
15335 for `sysconf()`.

- 15336 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for  
15337 the *symlink()* function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.
- 15338 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding `_PC_2_SYMLINKS`  
15339 to the symbolic constants list for *pathconf()*. This corresponds to the definition of  
15340 `POSIX2_SYMLINKS` in the Shell and Utilities volume of IEEE Std 1003.1-2001.
- 15341 **Issue 7**
- 15342 Austin Group Interpretations 1003.1-2001 #026 and #047 are applied.
- 15343 Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to  
15344 enable threads.
- 15345 Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for  
15346 `_POSIX2_CHAR_TERM`.
- 15347 SD5-XBD-ERN-41 is applied, adding the `_POSIX2_SYMLINKS` constant.
- 15348 SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied.
- 15349 Symbols to support the UUCP Utilities option are added.
- 15350 The variables for the supported programming environments are updated to be V7.
- 15351 The LEGACY and obsolescent symbols are removed.
- 15352 The *faccessat()*, *fchownat()*, *fexecve()*, *linkat()*, *readlinkat()*, *symlinkat()*, and *unlinkat()* functions  
15353 are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 15354 The `_POSIX_TRACE*` constants from the Trace option are marked obsolescent.
- 15355 The `_POSIX2_PBS*` constants from the Batch Environment Services and Utilities option are  
15356 marked obsolescent.
- 15357 Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory  
15358 Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks,  
15359 Threads, Timeouts, and Timers options is moved to the Base.
- 15360 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options  
15361 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex  
15362 or Robust Mutex Priority Inheritance, respectively.
- 15363 This reference page is clarified with respect to macros and symbolic constants.
- 15364 Changes are made related to support for finegrained timestamps and the  
15365 `_POSIX_TIMESTAMP_RESOLUTION` constant is added.
- 15366 The `_SC_THREAD_ROBUST_PRIO_INHERIT` and `_SC_THREAD_ROBUST_PRIO_PROTECT`  
15367 symbolic constants are added.

15368 **NAME**15369 `utime.h` — access and modification times structure15370 **SYNOPSIS**15371 OB `#include <utime.h>`15372 **DESCRIPTION**15373 The <utime.h> header shall declare the **utimbuf** structure, which shall include the following  
15374 members:15375 `time_t` `actime` Access time.  
15376 `time_t` `modtime` Modification time.

15377 The times shall be measured in seconds since the Epoch.

15378 The <utime.h> header shall define the **time\_t** type as described in <sys/types.h>.15379 The following shall be declared as a function and may also be defined as a macro. A function  
15380 prototype shall be provided.15381 `int utime(const char *, const struct utimbuf *)`;15382 **APPLICATION USAGE**15383 The `utime()` function only allows setting file timestamps to the nearest second. Applications  
15384 should use the `utimensat()` function instead. See <sys/stat.h>.15385 **RATIONALE**

15386 None.

15387 **FUTURE DIRECTIONS**

15388 The &lt;utime.h&gt; header may be removed in a future version.

15389 **SEE ALSO**

15390 &lt;sys/stat.h&gt;, &lt;sys/types.h&gt;

15391 XSH `futimens()`, `utime()`15392 **CHANGE HISTORY**

15393 First released in Issue 3.

15394 **Issue 6**15395 The following new requirements on POSIX implementations derive from alignment with the  
15396 Single UNIX Specification:

- 15397
- The **time\_t** type is defined.

15398 **Issue 7**

15399 The &lt;utime.h&gt; header is marked obsolescent.

**<utmpx.h>**15400 **NAME**

15401 utmpx.h — user accounting database definitions

15402 **SYNOPSIS**15403 XSI `#include <utmpx.h>`15404 **DESCRIPTION**15405 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following  
15406 members:

15407	char	ut_user[]	User login name.
15408	char	ut_id[]	Unspecified initialization process identifier.
15409	char	ut_line[]	Device name.
15410	pid_t	ut_pid	Process ID.
15411	short	ut_type	Type of entry.
15412	struct timeval	ut_tv	Time entry was made.

15413 The **<utmpx.h>** header shall define the **pid\_t** type through **typedef**, as described in  
15414 **<sys/types.h>**.15415 The **<utmpx.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.15416 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.15417 The **<utmpx.h>** header shall define the following symbolic constants as possible values for the  
15418 *ut\_type* member of the **utmpx** structure:

15419	EMPTY	No valid user accounting information.
15420	BOOT_TIME	Identifies time of system boot.
15421	OLD_TIME	Identifies time when system clock changed.
15422	NEW_TIME	Identifies time after system clock changed.
15423	USER_PROCESS	Identifies a process.
15424	INIT_PROCESS	Identifies a process spawned by the init process.
15425	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
15426	DEAD_PROCESS	Identifies a session leader who has exited.

15427 The following shall be declared as functions and may also be defined as macros. Function  
15428 prototypes shall be provided.

```

15429 void          endutxent (void);
15430 struct utmpx *getutxent (void);
15431 struct utmpx *getutxid (const struct utmpx *);
15432 struct utmpx *getutxline (const struct utmpx *);
15433 struct utmpx *pututxline (const struct utmpx *);
15434 void          setutxent (void);

```

*Headers***<utmpx.h>**

- 15435 **APPLICATION USAGE**  
15436       None.
- 15437 **RATIONALE**  
15438       None.
- 15439 **FUTURE DIRECTIONS**  
15440       None.
- 15441 **SEE ALSO**  
15442       [<sys/time.h>](#), [<sys/types.h>](#)  
15443       XSH *endutxent()*
- 15444 **CHANGE HISTORY**  
15445       First released in Issue 4, Version 2.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**<wchar.h>**15446 **NAME**15447            **wchar.h** — wide-character handling15448 **SYNOPSIS**

15449            #include &lt;wchar.h&gt;

15450 **DESCRIPTION**

15451 CX        Some of the functionality described on this reference page extends the ISO C standard.  
 15452        Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 15453        enable the visibility of these symbols in this header.

15454        The &lt;wchar.h&gt; header shall define the following types:

15455 CX        **FILE**            As described in <stdio.h>.15456 CX        **locale\_t**        As described in <locale.h>.

15457        **mbstate\_t**        An object type other than an array type that can hold the conversion state  
 15458        information necessary to convert between sequences of (possibly multi-byte)  
 15459 CX        characters and wide characters. If a codeset is being used such that an  
 15460        **mbstate\_t** needs to preserve more than two levels of reserved state, the results  
 15461        are unspecified.

15462        **size\_t**            As described in <stddef.h>.15463 CX        **va\_list**        As described in <stdarg.h>.15464        **wchar\_t**           As described in <stddef.h>.

15465 OB XSI    **wctype\_t**        A scalar type of a data object that can hold values which represent locale-  
 15466        specific character classification.

15467        **wint\_t**            An integer type capable of storing any valid value of **wchar\_t** or WEOF.

15468        The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are  
 15469        described in the <time.h> header.

15470        The implementation shall support one or more programming environments in which the width  
 15471        of **wint\_t** is no greater than the width of type **long**. The names of these programming  
 15472        environments can be obtained using the *confstr()* function or the *getconf* utility.

15473        The &lt;wchar.h&gt; header shall define the following macros:

15474        **WCHAR\_MAX**        As described in <stdint.h>.15475        **WCHAR\_MIN**        As described in <stdint.h>.

15476        **WEOF**            Constant expression of type **wint\_t** that is returned by several WP functions to  
 15477        indicate end-of-file.

15478        **NULL**            As described in <stddef.h>.

15479 CX        Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,  
 15480        <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>.

15481        The following shall be declared as functions and may also be defined as macros. Function  
 15482        prototypes shall be provided for use with ISO C standard compilers.

15483        **wint\_t**            btowc(int);15484        **wint\_t**            fgetwc(FILE \*);15485        **wchar\_t**        \*fgetws(wchar\_t \*restrict, int, FILE \*restrict);15486        **wint\_t**            fputwc(wchar\_t, FILE \*);15487        **int**                fputws(const wchar\_t \*restrict, FILE \*restrict);

## Headers

## &lt;wchar.h&gt;

```

15488     int             fwide(FILE *, int);
15489     int             fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15490     int             fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15491     wint_t          getwc(FILE *);
15492     wint_t          getwchar(void);
15493     OB XSI         int             iswalnum(wint_t);
15494     int             iswalpha(wint_t);
15495     int             iswcntrl(wint_t);
15496     int             iswctype(wint_t, wctype_t);
15497     int             iswdigit(wint_t);
15498     int             iswgraph(wint_t);
15499     int             iswlower(wint_t);
15500     int             iswprint(wint_t);
15501     int             iswpunct(wint_t);
15502     int             iswspace(wint_t);
15503     int             iswupper(wint_t);
15504     int             iswxdigit(wint_t);
15505     size_t          mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15506     size_t          mbrtowc(wchar_t *restrict, const char *restrict, size_t,
15507                             mbstate_t *restrict);
15508     int             mbsinit(const mbstate_t *);
15509     CX             size_t          mbsnrtowcs(wchar_t *restrict, const char **restrict,
15510                                             size_t, size_t, mbstate_t *restrict);
15511     size_t          mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15512                             mbstate_t *restrict);
15513     CX             FILE            *open_wmemstream(wchar_t **, size_t *);
15514     wint_t          putwc(wchar_t, FILE *);
15515     wint_t          putwchar(wchar_t);
15516     int             swprintf(wchar_t *restrict, size_t,
15517                             const wchar_t *restrict, ...);
15518     int             swscanf(const wchar_t *restrict,
15519                             const wchar_t *restrict, ...);
15520     OB XSI         wint_t          tolower(wint_t);
15521     wint_t          toupper(wint_t);
15522     wint_t          ungetwc(wint_t, FILE *);
15523     int             vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15524     int             vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15525     int             vswprintf(wchar_t *restrict, size_t,
15526                             const wchar_t *restrict, va_list);
15527     int             vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15528                             va_list);
15529     int             vwprintf(const wchar_t *restrict, va_list);
15530     int             vwscanf(const wchar_t *restrict, va_list);
15531     CX             wchar_t          *wcpcpy(wchar_t restrict*, const wchar_t *restrict);
15532     wchar_t          *wcpncpy(wchar_t restrict *, const wchar_t *restrict, size_t);
15533     size_t          wcrtoomb(char *restrict, wchar_t, mbstate_t *restrict);
15534     CX             int             wcscasecmp(const wchar_t *, const wchar_t *);
15535     int             wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
15536     wchar_t          *wcscat(wchar_t *restrict, const wchar_t *restrict);
15537     wchar_t          *wcschr(const wchar_t *, wchar_t);
15538     int             wcscmp(const wchar_t *, const wchar_t *);
15539     int             wcscoll(const wchar_t *, const wchar_t *);

```

## &lt;wchar.h&gt;

## Headers

15540	CX	int	wscoll_l(const wchar_t *, const wchar_t *, locale_t);
15541		wchar_t	*wcscpy(wchar_t *restrict, const wchar_t *restrict);
15542		size_t	wcscspn(const wchar_t *, const wchar_t *);
15543	CX	wchar_t	*wcsdup(const wchar_t *);
15544		size_t	wcsftime(wchar_t *restrict, size_t,
15545			const wchar_t *restrict, const struct tm *restrict);
15546		size_t	wcslen(const wchar_t *);
15547	CX	int	wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
15548		int	wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,
15549			locale_t);
15550		wchar_t	*wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15551		int	wcsncmp(const wchar_t *, const wchar_t *, size_t);
15552		wchar_t	*wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15553	CX	size_t	wcsnlen(const wchar_t *, size_t);
15554		size_t	wcsnrtombs(char *restrict, const wchar_t **restrict, size_t,
15555			size_t, mbstate_t *restrict);
15556		wchar_t	*wcpbrk(const wchar_t *, const wchar_t *);
15557		wchar_t	*wcsrchr(const wchar_t *, wchar_t);
15558		size_t	wcsrtombs(char *restrict, const wchar_t **restrict,
15559			size_t, mbstate_t *restrict);
15560		size_t	wcsspn(const wchar_t *, const wchar_t *);
15561		wchar_t	*wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15562		double	wctod(const wchar_t *restrict, wchar_t **restrict);
15563		float	wctof(const wchar_t *restrict, wchar_t **restrict);
15564		wchar_t	*wcstok(wchar_t *restrict, const wchar_t *restrict,
15565			wchar_t **restrict);
15566		long	wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15567		long double	wcstold(const wchar_t *restrict, wchar_t **restrict);
15568		long long	wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15569		unsigned long	wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15570		unsigned long long	
15571			wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15572	XSI	int	wcswidth(const wchar_t *, size_t);
15573		size_t	wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15574	CX	size_t	wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
15575			size_t, locale_t);
15576		int	wctob(wint_t);
15577	OB XSI	wctype_t	wctype(const char *);
15578	XSI	int	wcwidth(wchar_t);
15579		wchar_t	*wmemchr(const wchar_t *, wchar_t, size_t);
15580		int	wmemcmp(const wchar_t *, const wchar_t *, size_t);
15581		wchar_t	*wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15582		wchar_t	*wmemmove(wchar_t *, const wchar_t *, size_t);
15583		wchar_t	*wmemset(wchar_t *, wchar_t, size_t);
15584		int	wprintf(const wchar_t *restrict, ...);
15585		int	wscanf(const wchar_t *restrict, ...);

15586 **APPLICATION USAGE**

15587 The *iswblank()* function was a late addition to the ISO C standard and was introduced at the  
 15588 same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw\*()*  
 15589 functions. The Open Group Base Specifications had previously aligned with the MSE working  
 15590 draft and had introduced the rest of the *isw\*()* functions into <wchar.h>. For backwards-  
 15591 compatibility, the original set of *isw\*()* functions, without *iswblank()*, are permitted (as part of  
 15592 the XSI option) in <wchar.h>. For maximum portability, applications should include  
 15593 <wctype.h> in order to obtain declarations for the *isw\*()* functions. This compatibility has been  
 15594 made obsolescent.

15595 **RATIONALE**

15596 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their  
 15597 presence here is thus an extension.

15598 **FUTURE DIRECTIONS**

15599 None.

15600 **SEE ALSO**

15601 <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdint.h>, <stdio.h>, <stdlib.h>, <string.h>,  
 15602 <time.h>, <wctype.h>

15603 XSH Section 2.2 (on page 468), *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwide()*,  
 15604 *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*,  
 15605 *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mbrlen()*,  
 15606 *mbrtowc()*, *mbsinit()*, *mbsrtowcs()*, *open\_memstream()*, *putwc()*, *putwchar()*, *towlower()*,  
 15607 *towupper()*, *ungetwc()*, *vwfprintf()*, *vwscanf()*, *wrtomb()*, *wcscasecmp()*, *wcscat()*, *wcschr()*,  
 15608 *wcscmp()*, *wcscoll()*, *wcscpy()*, *wcscspn()*, *wcsdup()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*,  
 15609 *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsrtombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstok()*, *wcstol()*,  
 15610 *wcstoul()*, *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*,  
 15611 *wmemcpy()*, *wmemmove()*, *wmemset()*

15612 XCU *getconf*

15613 **CHANGE HISTORY**

15614 First released in Issue 4.

15615 **Issue 5**

15616 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15617 **Issue 6**

15618 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and  
 15619 *wcwidth()* are marked as extensions.

15620 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*  
 15621 function.

15622 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15623 • Various function prototypes are updated to add the **restrict** keyword.
- 15624 • The functions *vwscanf()*, *vwscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are  
 15625 added.

15626 The type **wctype\_t**, the *isw\*()*, *to\*()*, and *wctype()* functions are marked as XSI extensions.

15627 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION  
 15628 USAGE section.

## &lt;wchar.h&gt;

Headers

15629 **Issue 7**

15630 The *mbsnrto wcs()*, *open\_wmemstream()*, *wcpcpy()*, *wcpncpy()*, *wscasecmp()*, *wcsdup()*,  
15631 *wcsncasecmp()*, *wcsnlen()*, and *wscnrto mbs()* functions are added from The Open Group  
15632 Technical Standard, 2006, Extended API Set Part 1.

15633 The *wscasecmp\_l()*, *wcsncasecmp\_l()*, *wscoll\_l()*, and *wcsxfrm\_l()* functions are added from The  
15634 Open Group Technical Standard, 2006, Extended API Set Part 4.

15635 The **wctype\_t** type, and the *isw\**, *towlower()*, and *towupper()* functions are marked obsolescent in  
15636 <wchar.h> since the ISO C standard requires the declarations to be in <wctype.h>.

15637 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
15638 for the **locale\_t** type is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Headers

## &lt;wctype.h&gt;

## 15639 NAME

15640 wctype.h — wide-character classification and mapping utilities

## 15641 SYNOPSIS

15642 #include <wctype.h>

## 15643 DESCRIPTION

15644 CX Some of the functionality described on this reference page extends the ISO C standard.  
 15645 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to  
 15646 enable the visibility of these symbols in this header.

15647 The <wctype.h> header shall define the following types:

15648 **wint\_t** As described in <wchar.h>.

15649 **wctrans\_t** A scalar type that can hold values which represent locale-specific character  
 15650 mappings.

15651 **wctype\_t** As described in <wchar.h>.

15652 CX The <wctype.h> header shall define the **locale\_t** type as described in <locale.h>.

15653 The <wctype.h> header shall define the following macro:

15654 **WEOF** As described in <wchar.h>.

15655 For all functions described in this header that accept an argument of type **wint\_t**, the value is  
 15656 representable as a **wchar\_t** or equals the value of **WEOF**. If this argument has any other value,  
 15657 the behavior is undefined.

15658 The behavior of these functions shall be affected by the *LC\_CTYPE* category of the current locale.

15659 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,  
 15660 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

15661 The following shall be declared as functions and may also be defined as macros. Function  
 15662 prototypes shall be provided for use with ISO C standard compilers.

```

15663 int iswalnum(wint_t);
15664 CX int iswalnum_l(wint_t, locale_t);
15665 int iswalpna(wint_t);
15666 CX int iswalpna_l(wint_t, locale_t);
15667 int iswblank(wint_t);
15668 CX int iswblank_l(wint_t, locale_t);
15669 int iswcntrl(wint_t);
15670 CX int iswcntrl_l(wint_t, locale_t);
15671 int iswctype(wint_t, wctype_t);
15672 CX int iswctype_l(wint_t, wctype_t, locale_t);
15673 int iswdigit(wint_t);
15674 CX int iswdigit_l(wint_t, locale_t);
15675 int iswgraph(wint_t);
15676 CX int iswgraph_l(wint_t, locale_t);
15677 int iswlower(wint_t);
15678 CX int iswlower_l(wint_t, locale_t);
15679 int iswprint(wint_t);
15680 CX int iswprint_l(wint_t, locale_t);
15681 int iswpunct(wint_t);
15682 CX int iswpunct_l(wint_t, locale_t);
15683 int iswspace(wint_t);
  
```

## &lt;wctype.h&gt;

Headers

```

15684 CX      int      iswspace_l(wint_t, locale_t);
15685        int      iswupper(wint_t);
15686 CX      int      iswupper_l(wint_t, locale_t);
15687        int      iswxdigit(wint_t);
15688 CX      int      iswxdigit_l(wint_t, locale_t);
15689        wint_t    towctrans(wint_t, wctrans_t);
15690 CX      wint_t    towctrans_l(wint_t, wctrans_t, locale_t);
15691        wint_t    tolower(wint_t);
15692 CX      wint_t    tolower_l(wint_t, locale_t);
15693        wint_t    toupper(wint_t);
15694 CX      wint_t    toupper_l(wint_t, locale_t);
15695        wctrans_t wctrans(const char *);
15696 CX      wctrans_t wctrans_l(const char *, locale_t);
15697        wctype_t  wctype(const char *);
15698 CX      wctype_t  wctype_l(const char *, locale_t);

```

15699 **APPLICATION USAGE**

15700 None.

15701 **RATIONALE**

15702 None.

15703 **FUTURE DIRECTIONS**

15704 None.

15705 **SEE ALSO**

15706 [<ctype.h>](#), [<locale.h>](#), [<stdarg.h>](#), [<stddef.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<string.h>](#), [<time.h>](#),  
15707 [<wchar.h>](#)

15708 XSH Section 2.2 (on page 468), [iswalnum\(\)](#), [iswalpha\(\)](#), [iswblank\(\)](#), [iswcntrl\(\)](#), [iswctype\(\)](#),  
15709 [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#), [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#),  
15710 [setlocale\(\)](#), [towctrans\(\)](#), [tolower\\_l\(\)](#), [toupper\\_l\(\)](#), [wctrans\(\)](#), [wctype\(\)](#)

15711 **CHANGE HISTORY**

15712 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15713 **Issue 6**15714 The [iswblank\(\)](#) function is added for alignment with the ISO/IEC 9899:1999 standard.15715 **Issue 7**

15716 SD5-XBD-ERN-6 is applied.

15717 The [\\*\\_l\(\)](#) functions are added from The Open Group Technical Standard, 2006, Extended API Set  
15718 Part 4.

15719 This reference page is clarified with respect to macros and symbolic constants.

## Headers

## &lt;wordexp.h&gt;

## 15720 NAME

15721 wordexp.h — word-expansion types

## 15722 SYNOPSIS

15723 #include &lt;wordexp.h&gt;

## 15724 DESCRIPTION

15725 The <wordexp.h> header shall define the structures and symbolic constants used by the  
15726 *wordexp()* and *wordfree()* functions.15727 The <wordexp.h> header shall define the **wordexp\_t** structure type, which shall include at least  
15728 the following members:

15729	size_t	we_wordc	Count of words matched by <i>words</i> .
15730	char	**we_wordv	Pointer to list of expanded words.
15731	size_t	we_offs	Slots to reserve at the beginning of <i>we_wordv</i> .

15732 The <wordexp.h> header shall define the following symbolic constants for use as flags for the  
15733 *wordexp()* function:

15734	WRDE_APPEND	Append words to those previously generated.
15735	WRDE_DOOFFS	Number of null pointers to prepend to <i>we_wordv</i> .
15736	WRDE_NOCMD	Fail if command substitution is requested.
15737	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to 15738 <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result is the same 15739 as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> 15740 without WRDE_REUSE.

15741 WRDE\_SHOWERR Do not redirect *stderr* to */dev/null*.

15742 WRDE\_UNDEF Report error on an attempt to expand an undefined shell variable.

15743 The &lt;wordexp.h&gt; header shall define the following symbolic constants as error return values:

15744	WRDE_BADCHAR	One of the unquoted characters—<newline>, '   ', ' & ', ' ; ', ' < ', ' > ', 15745 ' ( ', ' ) ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
15746	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
15747	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
15748	WRDE_NOSPACE	Attempt to allocate memory failed.
15749	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated 15750 string.

15751 The <wordexp.h> header shall define the **size\_t** type as described in <stddef.h>.15752 The following shall be declared as functions and may also be defined as macros. Function  
15753 prototypes shall be provided.

```
15754 int wordexp(const char *restrict, wordexp_t *restrict, int);
15755 void wordfree(wordexp_t *);
```

## &lt;wordexp.h&gt;

Headers

15756 **APPLICATION USAGE**

15757 None.

15758 **RATIONALE**

15759 None.

15760 **FUTURE DIRECTIONS**

15761 None.

15762 **SEE ALSO**15763 [<stddef.h>](#)15764 XSH [Section 2.6](#)15765 **CHANGE HISTORY**

15766 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15767 **Issue 6**15768 The **restrict** keyword is added to the prototype for *wordexp()*.

15769 The WRDE\_NOSYS constant is marked obsolescent.

15770 **Issue 7**

15771 The obsolescent WRDE\_NOSYS constant is removed.

15772 This reference page is clarified with respect to macros and symbolic constants.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

15773

 *Technical Standard*

15774

**Vol. 2:**

15775

**System Interfaces, Issue 7**

15776

*The Open Group*

15777

*The Institute of Electrical and Electronics Engineers, Inc.*

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

15778

Chapter 1

15779

# Introduction

15780

The System Interfaces volume of POSIX.1-2008 describes the interfaces offered to application programs by POSIX-conformant systems.

15781

## 1.1 Relationship to Other Formal Standards

15783

Great care has been taken to ensure that this volume of POSIX.1-2008 is fully aligned with the following standards:

15784

15785

ISO C (1999)

15786

ISO/IEC 9899:1999, Programming Languages C, including ISO/IEC

15787

9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC

15788

9899:1999/Cor.3.

15789

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of POSIX.1-2008. Some functions and headers included within this volume of POSIX.1-2008 have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see [Section 1.7.1](#), on page 7). Any conflict between this volume of POSIX.1-2008 and the ISO C standard is unintentional.

15790

15791

15792

15793

15794

15795

This volume of POSIX.1-2008 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

15796

## 1.2 Format of Entries

15798

The entries in [Chapter 3](#) are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

15799

15800

### NAME

15801

This section gives the name or names of the entry and briefly states its purpose.

15802

### SYNOPSIS

15803

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

15804

15805

15806

```
#include <stdio.h>
```

15807

### DESCRIPTION

15808

This section describes the functionality of the function or header.

15809

### RETURN VALUE

15810

This section indicates the possible return values, if any.

15811

If the implementation can detect errors, "successful completion" means that no error

- 15812 has been detected during execution of the function. If the implementation does detect  
15813 an error, the error is indicated.
- 15814 For functions where no errors are defined, “successful completion” means that if the  
15815 implementation checks for errors, no error has been detected. If the implementation can  
15816 detect errors, and an error is detected, the indicated return value is returned and *errno*  
15817 may be set.
- 15818 **ERRORS**
- 15819 This section gives the symbolic names of the error values returned by a function or  
15820 stored into a variable accessed through the symbol *errno* if an error occurs.
- 15821 “No errors are defined” means that error values returned by a function or stored into a  
15822 variable accessed through the symbol *errno*, if any, depend on the implementation.
- 15823 **EXAMPLES**
- 15824 This section is informative.
- 15825 This section gives examples of usage, where appropriate. In the event of conflict  
15826 between an example and a normative part of this volume of POSIX.1-2008, the  
15827 normative material is to be taken as correct.
- 15828 **APPLICATION USAGE**
- 15829 This section is informative.
- 15830 This section gives warnings and advice to application developers about the entry. In the  
15831 event of conflict between warnings and advice and a normative part of this volume of  
15832 POSIX.1-2008, the normative material is to be taken as correct.
- 15833 **RATIONALE**
- 15834 This section is informative.
- 15835 This section contains historical information concerning the contents of this volume of  
15836 POSIX.1-2008 and why features were included or discarded by the standard  
15837 developers.
- 15838 **FUTURE DIRECTIONS**
- 15839 This section is informative.
- 15840 This section provides comments which should be used as a guide to current thinking;  
15841 there is not necessarily a commitment to adopt these future directions.
- 15842 **SEE ALSO**
- 15843 This section is informative.
- 15844 This section gives references to related information.
- 15845 **CHANGE HISTORY**
- 15846 This section is informative.
- 15847 This section shows the derivation of the entry and any significant changes that have  
15848 been made to it.

15849

Chapter 2

15850

## General Information

15851

This chapter covers information that is relevant to all the functions specified in Chapter 3 and XBD Chapter 13 (on page 219).

15852

## 2.1 Use and Implementation of Interfaces

15853

### 2.1.1 Use and Implementation of Functions

15854

15855

Each of the following statements shall apply to all functions unless explicitly stated otherwise in the detailed descriptions that follow:

15856

15857

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.

15858

15859

15860

15861

15862

15863

15864

15865

15866

15867

2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the <left-parenthesis> that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition shall also ensure that an actual function is referred to.

15868

15869

15870

3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments.

15871

15872

15873

4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.

15874

15875

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

15876 **2.1.2 Use and Implementation of Macros**

15877 Each of the following statements shall apply to all macros unless explicitly stated otherwise:

- 15878 1. Any definition of an object-like macro in a header shall expand to code that is fully  
15879 protected by parentheses where necessary, so that it groups in an arbitrary expression as  
15880 if it were a single identifier.
- 15881 2. All object-like macros listed as expanding to integer constant expressions shall  
15882 additionally be suitable for use in `#if` preprocessing directives.
- 15883 3. Any definition of a function-like macro in a header shall expand to code that evaluates  
15884 each of its arguments exactly once, fully protected by parentheses where necessary, so  
15885 that it is generally safe to use arbitrary expressions as arguments.
- 15886 4. Any definition of a function-like macro in a header can be invoked in an expression  
15887 anywhere a function with a compatible return type could be called.

15888 **2.2 The Compilation Environment**15889 **2.2.1 POSIX.1 Symbols**

15890 Certain symbols in this volume of POSIX.1-2008 are defined in headers (see XBD [Chapter 13](#), on  
15891 page 219). Some of those headers could also define symbols other than those defined by  
15892 POSIX.1-2008, potentially conflicting with symbols used by the application. Also, POSIX.1-2008  
15893 defines symbols that are not permitted by other standards to appear in those headers without  
15894 some control on the visibility of those symbols.

15895 Symbols called “feature test macros” are used to control the visibility of symbols that might be  
15896 included in a header. Implementations, future versions of this standard, and other standards  
15897 may define additional feature test macros.

15898 In the compilation of an application that `#defines` a feature test macro specified by  
15899 POSIX.1-2008, no header defined by POSIX.1-2008 shall be included prior to the definition of the  
15900 feature test macro. This restriction also applies to any implementation-provided header in  
15901 which these feature test macros are used. If the definition of the macro does not precede the  
15902 `#include`, the result is undefined.

15903 Feature test macros shall begin with the <underscore> character (‘\_’).

15904 **2.2.1.1 The `_POSIX_C_SOURCE` Feature Test Macro**

15905 A POSIX-conforming application shall ensure that the feature test macro `_POSIX_C_SOURCE` is  
15906 defined before inclusion of any header.

15907 When an application includes a header described by POSIX.1-2008, and when this feature test  
15908 macro is defined to have the value 200809L:

- 15909 1. All symbols required by POSIX.1-2008 to appear when the header is included shall be  
15910 made visible.

- 15911 2. Symbols that are explicitly permitted, but not required, by POSIX.1-2008 to appear in that  
15912 header (including those in reserved name spaces) may be made visible.
- 15913 3. Additional symbols not required or explicitly permitted by POSIX.1-2008 to be in that  
15914 header shall not be made visible, except when enabled by another feature test macro.
- 15915 Identifiers in POSIX.1-2008 may only be undefined using the **#undef** directive as described in  
15916 [Section 2.1](#) (on page 467) or [Section 2.2.2](#). These **#undef** directives shall follow all **#include**  
15917 directives of any header in POSIX.1-2008.
- 15918 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been  
15919 superseded by `_POSIX_C_SOURCE`.

#### 15920 2.2.1.2 The `_XOPEN_SOURCE` Feature Test Macro

15921 XSI An XSI-conforming application shall ensure that the feature test macro `_XOPEN_SOURCE` is  
15922 defined with the value 700 before inclusion of any header. This is needed to enable the  
15923 functionality described in [Section 2.2.1.1](#) (on page 468) and to ensure that the XSI option is  
15924 enabled.

15925 Since this volume of POSIX.1-2008 is aligned with the ISO C standard, and since all functionality  
15926 enabled by `_POSIX_C_SOURCE` set equal to 200809L is enabled by `_XOPEN_SOURCE` set equal  
15927 to 700, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so  
15928 defined. Therefore, if `_XOPEN_SOURCE` is set equal to 700 and `_POSIX_C_SOURCE` is set equal  
15929 to 200809L, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to  
15930 700. However, should `_POSIX_C_SOURCE` be set to a value greater than 200809L, the behavior  
15931 is unspecified.

15932 If `_XOPEN_SOURCE` is defined with the value 700 and `_POSIX_C_SOURCE` is undefined before  
15933 inclusion of any header, then the header may define the `_POSIX_C_SOURCE` macro with the  
15934 value 200809L.

### 15935 2.2.2 The Name Space

15936 All identifiers in this volume of POSIX.1-2008, except *environ*, are defined in at least one of the  
15937 XSI headers, as shown in XBD [Chapter 13](#) (on page 219). When `_XOPEN_SOURCE` or  
15938 `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially  
15939 conflicting with identifiers used by the application. The set of identifiers visible to the  
15940 application consists of precisely those identifiers from the header pages of the included headers,  
15941 as well as additional identifiers reserved for the implementation. In addition, some headers may  
15942 make visible identifiers from other headers as indicated on the relevant header pages.

15943 Implementations may also add members to a structure or union without controlling the  
15944 visibility of those members with a feature test macro, as long as a user-defined macro with the  
15945 same name cannot interfere with the correct interpretation of the program. The identifiers  
15946 reserved for use by the implementation are described below:

- 15947 1. Each identifier with external linkage described in the header section is reserved for use as  
15948 an identifier with external linkage if the header is included.
- 15949 2. Each macro described in the header section is reserved for any use if the header is  
15950 included.
- 15951 3. Each identifier with file scope described in the header section is reserved for use as an  
15952 identifier with file scope in the same name space if the header is included.

15953 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by POSIX.1-2008 and other

- 15954 POSIX standards. Implementations may add symbols to the headers shown in the following  
15955 table, provided the identifiers for those symbols either:
- 15956 1. Begin with the corresponding reserved prefixes in the table, or
  - 15957 2. Have one of the corresponding complete names in the table, or
  - 15958 3. End in the string indicated as a reserved suffix in the table and do not use the reserved  
15959 prefixes `posix_`, `POSIX_`, or `_POSIX_`, as long as the reserved suffix is in that part of the  
15960 name considered significant by the implementation.
- 15961 Symbols that use the reserved prefix `_POSIX_` may be made visible by implementations in any  
15962 header defined by POSIX.1-2008.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## General Information

## The Compilation Environment

	Header	Prefix	Suffix	Complete Name
15963				
15964				
15965	<aio.h>	aio_, lio_, AIO_, LIO_		
15966	<arpa/inet.h>	inet_		
15967	<ctype.h>	to[a-z], is[a-z]		
15968	<dlfcn.h>	RTLD_		
15969	<dirent.h>	d_		
15970	<fcntl.h>	l_		
15971 XSI	<fmtmsg.h>	MM_		
15972	<fnmatch.h>	FNM_		
15973 XSI	<ftw.h>	FTW		
15974	<glob.h>	gl_, GLOB_		
15975	<grp.h>	gr_		
15976	<limits.h>		_MAX, _MIN	
15977 MSG	<mqueue.h>	mq_, MQ_		
15978 XSI	<ndbm.h>	dbm_, DBM_		
15979	<netdb.h>	ai_, h_, n_, p_, s_		
15980	<net/if.h>	if_, IF_		
15981	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
15982 IP6		in6_, s6_, sin6_, IPV6_		
15983	<netinet/tcp.h>	TCP_		
15984	<nl_types.h>	NL_		
15985	<poll.h>	pd_, ph_, ps_, POLL		
15986	<pthread.h>	pthread_, PTHREAD_		
15987	<pwd.h>	pw_		
15988	<regex.h>	re_, rm_, REG_		
15989	<sched.h>	sched_, SCHED_		
15990	<semaphore.h>	sem_, SEM_		
15991	<signal.h>	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_,		
15992		ss_, sv_, SS_, TRAP_		
15993 XSI		POLL_		
15994 OB XSR	<stropts.h>	bi_, ic_, l_, sl_, str_, FLUSH[A-Z], I_, S_, SND[A-Z]		
15995				
15996	<stdint.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
15997				
15998	<stdlib.h>	str[a-z]		
15999	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
16000				
16001 XSI	<sys/ipc.h>	ipc_, IPC_		key, pad, seq
16002	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
16003				
16004 XSI	<sys/msg.h>	msg, MSG_[A-Z]		msg
16005 XSI	<sys/resource.h>	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
16006	<sys/select.h>	fd_, fds_, FD_		

	Header	Prefix	Suffix	Complete Name
16007				
16008				
16009 XSI	<sys/sem.h>	sem, SEM_		sem
16010 XSI	<sys/shm.h>	shm, SHM[A-Z], SHM_[A-Z]		
16011	<sys/socket.h>	cmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_, AF_, MSG_, PF_, SCM_, SHUT_, SO		
16012				
16013 XSI				
16014				
16015	<sys/stat.h>	st_		
16016	<sys/statvfs.h>	f_, ST_		
16017 XSI	<sys/time.h>	it_, tv_, ITIMER_		
16018	<sys/times.h>	tms_		
16019 XSI	<sys/uio.h>	iov_		UIO_MAXIOV
16020	<sys/un.h>	sun_		
16021	<sys/utsname.h>	uts_		
16022	<sys/wait.h>	P_, W[A-Z]		
16023 XSI	<syslog.h>	LOG_		
16024	<termios.h>	c_, B[0-9], TC		
16025	<time.h>	tm_ clock_, it_, timer_, tv_, CLOCK_, TIMER_		
16026				
16027				
16028 XSI	<ulimit.h>	UL_		
16029	<utime.h>	utim_		
16030 XSI	<utmpx.h>	ut_	_LVL, _PROCESS, _TIME	
16031				
16032	<wchar.h>	wcs[a-z]		
16033	<wctype.h>	is[a-z], to[a-z]		
16034	<wordexp.h>	we_, WRDE_		
16035	ANY header		_t	

**Note:** The notation [A-Z] indicates any uppercase letter in the portable character set. The notation [a-z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

16039 If any header in the following table is included, macros with the prefixes shown may be defined.  
 16040 After the last inclusion of a given header, an application may use identifiers with the  
 16041 corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the  
 16042 corresponding macro.

Header	Prefix
<errno.h>	E[0-9], E[A-Z]
<fcntl.h>	F_, O_
<fenv.h>	FE_[A-Z]
<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
<locale.h>	LC_[A-Z]
<math.h>	FP_[A-Z]
<netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_
16051 IP6	IN6_
16052	<signal.h> SIG_, SIG[A-Z],
16053 XSI	SV_
16054 CX	<stdio.h> SEEK_
16055 OB XSR	<stropts.h> M_, MUXID_R[A-Z], STR
16056 XSI	<sys/resource.h> RLIM_
16057 XSI	<sys/socket.h> CMSG_
16058	<sys/stat.h> S_
16059 XSI	<sys/uio.h> IOV_
16060	<termios.h> I, O, V (See below.)
16061	<unistd.h> SEEK_

16062 The following are used to reserve complete names for the <stdint.h> header:

16063 INT[0-9A-Za-z\_]\*\_MIN  
 16064 INT[0-9A-Za-z\_]\*\_MAX  
 16065 INT[0-9A-Za-z\_]\*\_C  
 16066 UINT[0-9A-Za-z\_]\*\_MIN  
 16067 UINT[0-9A-Za-z\_]\*\_MAX  
 16068 UINT[0-9A-Za-z\_]\*\_C

16069 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the  
 16070 portable character set. The notation [0-9a-z\_] indicates any digit, any lowercase letter in the  
 16071 portable character set, or <underscore>.

16072 XSI The following reserved names are used as exact matches for <termios.h>:

16073 CBAUD	EXTB	VDSUSP
16074 DEFECHO	FLUSHO	VLNEXT
16075 ECHOCTL	LOBLK	VREPRINT
16076 ECHOK	PENDIN	VSTATUS
16077 ECHOPRT	SWTCH	VWERASE
16078 EXTA	VDISCARD	

- 16079 The following identifiers are reserved regardless of the inclusion of headers:
- 16080 1. With the exception of identifiers beginning with the prefix `_POSIX_`, all identifiers that
- 16081 begin with an `<underscore>` and either an uppercase letter or another `<underscore>` are
- 16082 always reserved for any use by the implementation.
- 16083 2. All identifiers that begin with an `<underscore>` are always reserved for use as identifiers
- 16084 with file scope in both the ordinary identifier and tag name spaces.
- 16085 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
- 16086 Some of these identifiers do not appear in this volume of POSIX.1-2008, but are reserved for
- 16087 future use by the ISO C standard.
- 16088 4. All functions and external identifiers defined in XBD [Chapter 13](#) (on page 219) are reserved
- 16089 for use as identifiers with external linkage.
- 16090 5. All the identifiers defined in this volume of POSIX.1-2008 that have external linkage are
- 16091 always reserved for use as identifiers with external linkage.
- 16092 **Note:** The notation `[a-z]` indicates any lowercase letter in the portable character set. The notation `'**'`
- 16093 indicates any combination of digits, letters in the portable character set, or `<underscore>`.
- 16094 No other identifiers are reserved.

## General Information

## The Compilation Environment

16095	_Exit	casinh	clog10	erfcl	fminl
16096	abort	casinhf	clog10f	erff	fmod
16097	abs	casinhl	clog10l	erfl	fmodf
16098	acos	casinl	clog1p	errno	fmodl
16099	acosf	catan	clog1pf	exit	fopen
16100	acosh	catanf	clog1pl	exp	fprintf
16101	acoshf	catanh	clog2	exp2	fputc
16102	acoshl	catanh	clog2f	exp2f	fputs
16103	acosl	catanhf	clog2l	exp2l	fputwc
16104	acosl	catanhf	clogf	expf	fputws
16105	asctime	catanhl	clogl	expl	fread
16106	asin	catanhl	conj	expm1	free
16107	asinf	catanl	conjf	expm1f	freopen
16108	asinh	cbrt	conjl	expm1l	frexp
16109	asinhf	cbrtf	copysign	fabs	frexpf
16110	asinhf	cbrtl	copysignf	fabsf	frexpl
16111	asinl	ccos	copysignl	fabsl	fscanf
16112	asinl	ccosf	cos	fclose	fseek
16113	atan	ccosh	cosf	fdim	fsetpos
16114	atan2	ccoshf	cosh	fdimf	ftell
16115	atan2f	ccoshl	coshf	fdiml	fwide
16116	atan2l	ccosl	coshl	feclearexcept	fwprintf
16117	atanf	ceil	cosl	fegetenv	fwrite
16118	atanf	ceilf	cpow	fegetexceptflag	fwscanf
16119	atanh	ceilf	cpowf	fegetround	getc
16120	atanh	ceill	cpowl	fehldexcept	getchar
16121	atanhf	ceill	cproj	feof	getenv
16122	atanhl	cerf	cprojf	feraiseexcept	gets
16123	atanl	cerfc	cprojl	ferror	getwc
16124	atanl	cerfcf	creal	fesetenv	getwchar
16125	atexit	cerfcl	crealf	fesetexceptflag	gmtime
16126	atof	cerff	creall	fesetround	hypotf
16127	atoi	cerfl	csin	fetestexcept	hypotl
16128	atol	cexpm1	csinf	feupdateenv	ilogb
16129	atoll	cexpm1f	csinh	fflush	ilogbf
16130	bsearch	cexpm1l	csinhf	fgetc	ilogbl
16131	cabs	cexp	csinhf	fgetpos	imaxabs
16132	cabsf	cexp2	csinl	fgets	imaxdiv
16133	cabsl	cexp2f	csqrt	fgetwc	is[a-z]*
16134	cacos	cexp2l	csqrtf	fgetws	isblank
16135	cacosf	cexpf	csqrtl	floor	iswblank
16136	cacosh	cexpl	ctan	floorf	labs
16137	cacoshf	cimag	ctanf	floorl	ldexp
16138	cacoshl	cimagf	ctanl	fma	ldexpf
16139	cacosl	cimagl	ctgamma	fmaf	ldexpl
16140	calloc	clearerr	ctgammaf	fmaf	ldiv
16141	carg	clgamma	ctgamma	fmax	ldiv
16142	cargf	clgammaf	ltime	fmaxf	lgammaf
16143	cargl	clgamma	difftime	fmaxl	lgamma
16144	casin	clock	div	fmin	llabs
16145	casinf	clog	erfcf	fminf	llrint

16146	llrintf	mbsinit	realloc	sprintf	ungetwc
16147	llrintl	mbsrtowcs	remainderf	sqrt	va_end
16148	llround	mbstowcs	remainderl	sqrtf	vfprintf
16149	llroundf	mbtowc	remove	sqrtl	vfscanf
16150	llroundl	mem[a-z]*	remquo	srand	vwprintf
16151	localeconv	mktime	remquof	sscanf	vwscanf
16152	localtime	modf	remquol	str[a-z]*	vprintf
16153	log	modff	rename	strtof	vscanf
16154	log10	modfl	rewind	strtoimax	vsprintf
16155	log10f	nan	rint	strtol	vsscanf
16156	log10l	nanf	rintf	strtoll	vswprintf
16157	log1p	nanl	rintl	strtoull	vswscanf
16158	log1pf	nearbyint	round	strtoumax	vwprintf
16159	log1pl	nearbyintf	roundf	swprintf	vwscanf
16160	log2	nearbyintl	roundl	swscanf	wcrtomb
16161	log2f	nextafterf	scalbn	system	wcs[a-z]*
16162	log2l	nextafterl	scalblnf	tan	wcstof
16163	logb	nexttoward	scalblnl	tanf	wcstoimax
16164	logbf	nexttowardf	scalbn	tanh	wcstold
16165	logbl	nexttowardl	scalbnf	tanhf	wcstoll
16166	logf	perror	scalbnl	tanhf	wcstoull
16167	logl	pow	scanf	tanl	wcstoumax
16168	longjmp	powf	setbuf	tgamma	wctob
16169	lrint	powl	setjmp	tgammaf	wctomb
16170	lrintf	printf	setlocale	tgamma	wctrans
16171	lrintl	putc	setvbuf	time	wctype
16172	lround	putchar	signal	tmpfile	wcwidth
16173	lroundf	puts	sin	tmpnam	wmem[a-z]*
16174	lroundl	putwc	sinf	to[a-z]*	wprintf
16175	malloc	putwchar	sinh	trunc	wscanf
16176	mblen	qsort	sinhf	truncf	
16177	mbrlen	raise	sinhl	truncl	
16178	mbrtowc	rand	sinl	ungetc	

16179 Applications shall not declare or define identifiers with the same name as an identifier reserved  
 16180 in the same context. Since macro names are replaced whenever found, independent of scope and  
 16181 name space, macro names matching any of the reserved identifier names shall not be defined by  
 16182 an application if any associated header is included.

16183 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,  
 16184 headers may be included in any order, and each may be included more than once in a given  
 16185 scope, with no difference in effect from that of being included only once.

16186 If used, the application shall ensure that a header is included outside of any external declaration  
 16187 or definition, and it shall be first included before the first reference to any type or macro it  
 16188 defines, or to any function or object it declares. However, if an identifier is declared or defined in  
 16189 more than one header, the second and subsequent associated headers may be included after the  
 16190 initial reference to the identifier. Prior to the inclusion of a header, the application shall not  
 16191 define any macros with names lexically identical to symbols defined by that header.

## 16192 2.3 Error Numbers

16193 Most functions can provide an error number. The means by which each function provides its  
16194 error numbers is specified in its description.

16195 Some functions provide the error number in a variable accessed through the symbol *errno*,  
16196 defined by including the `<errno.h>` header. The value of *errno* should only be examined when it  
16197 is indicated to be valid by a function's return value. No function in this volume of POSIX.1-2008  
16198 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by  
16199 function calls or assignments to *errno* by other threads.

16200 Some functions return an error number directly as the function value. These functions return a  
16201 value of zero to indicate success.

16202 If more than one error occurs in processing a function call, any one of the possible errors may be  
16203 returned, as the order of detection is undefined.

16204 Implementations may support additional errors not included in this list; may generate errors  
16205 included in this list under circumstances other than those described here, or may contain  
16206 extensions or limitations that prevent some errors from occurring.

16207 The ERRORS section on each reference page specifies which error conditions shall be detected  
16208 by all implementations ("shall fail") and which may be optionally detected by an  
16209 implementation ("may fail"). If no error condition is detected, the action requested shall be  
16210 successful.

16211 Implementations may generate error numbers listed here under circumstances other than those  
16212 described, if and only if all those error conditions can always be treated identically to the error  
16213 conditions as described in this volume of POSIX.1-2008. Implementations shall not generate a  
16214 different error number from one required by this volume of POSIX.1-2008 for an error condition  
16215 described in this volume of POSIX.1-2008, but may generate additional errors unless explicitly  
16216 disallowed for a particular function.

16217 Each implementation shall document, in the conformance document, situations in which each of  
16218 the optional conditions defined in POSIX.1-2008 is detected. The conformance document may  
16219 also contain statements that one or more of the optional error conditions are not detected.

16220 Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this  
16221 applies it is stated in the ERRORS section on the individual function pages.

16222 The following macro names identify the possible error numbers, in the context of the functions  
16223 specifically defined in this volume of POSIX.1-2008; these general descriptions are more  
16224 precisely defined in the ERRORS sections of the functions that return them. Only these macro  
16225 names should be used in programs, since the actual value of the error number is unspecified. All  
16226 values listed in this section shall be unique, except as noted below. The values for all these  
16227 macros shall be found in the `<errno.h>` header defined in the Base Definitions volume of  
16228 POSIX.1-2008. The actual values are unspecified by this volume of POSIX.1-2008.

16229 [E2BIG]

16230 Argument list too long. The sum of the number of bytes used by the new process image's  
16231 argument list and environment list is greater than the system-imposed limit of {ARG\_MAX}  
16232 bytes.

16233 or:

16234 Lack of space in an output buffer.

16235 or:

16236 Argument is greater than the system-imposed maximum.

16237	[EACCES]	
16238		Permission denied. An attempt was made to access a file in a way forbidden by its file
16239		access permissions.
16240	[EADDRINUSE]	
16241		Address in use. The specified address is in use.
16242	[EADDRNOTAVAIL]	
16243		Address not available. The specified address is not available from the local system.
16244	[EAFNOSUPPORT]	
16245		Address family not supported. The implementation does not support the specified address
16246		family, or the specified address is not a valid address for the address family of the specified
16247		socket.
16248	[EAGAIN]	
16249		Resource temporarily unavailable. This is a temporary condition and later calls to the same
16250		routine may complete normally.
16251	[EALREADY]	
16252		Connection already in progress. A connection request is already in progress for the specified
16253		socket.
16254	[EBADF]	
16255		Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
16256		read (write) request is made to a file that is only open for writing (reading).
16257	[EBADMSG]	
16258	OB XSR	Bad message. During a <i>read()</i> , <i>getmsg()</i> , <i>getpmsg()</i> , or <i>ioctl()</i> I_RECVFD request to a
16259		STREAMS device, a message arrived at the head of the STREAM that is inappropriate for
16260		the function receiving the message.
16261		<i>read()</i> Message waiting to be read on a STREAM is not a data message.
16262		<i>getmsg()</i> or <i>getpmsg()</i>
16263		A file descriptor was received instead of a control message.
16264		<i>ioctl()</i> Control or data information was received instead of a file descriptor when
16265		I_RECVFD was specified.
16266		or:
16267		Bad Message. The implementation has detected a corrupted message.
16268	[EBUSY]	
16269		Resource busy. An attempt was made to make use of a system resource that is not currently
16270		available, as it is being used by another process in a manner that would have conflicted
16271		with the request being made by this process.
16272	[ECANCELED]	
16273		Operation canceled. The associated asynchronous operation was canceled before
16274		completion.
16275	[ECHILD]	
16276		No child process. A <i>wait()</i> , <i>waitid()</i> , or <i>waitpid()</i> function was executed by a process that
16277		had no existing or unwaited-for child process.
16278	[ECONNABORTED]	
16279		Connection aborted. The connection has been aborted.

## General Information

## Error Numbers

16280	[ECONNREFUSED]
16281	Connection refused. An attempt to connect to a socket was refused because there was no
16282	process listening or because the queue of connection requests was full and the underlying
16283	protocol does not support retransmissions.
16284	[ECONNRESET]
16285	Connection reset. The connection was forcibly closed by the peer.
16286	[EDEADLK]
16287	Resource deadlock would occur. An attempt was made to lock a system resource that would
16288	have resulted in a deadlock situation.
16289	[EDESTADDRREQ]
16290	Destination address required. No bind address was established.
16291	[EDOM]
16292	Domain error. An input argument is outside the defined domain of the mathematical
16293	function (defined in the ISO C standard).
16294	[EDQUOT]
16295	Reserved.
16296	[EEXIST]
16297	File exists. An existing file was mentioned in an inappropriate context; for example, as a
16298	new link name in the <i>link()</i> function.
16299	[EFAULT]
16300	Bad address. The system detected an invalid address in attempting to use an argument of a
16301	call. The reliable detection of this error cannot be guaranteed, and when not detected may
16302	result in the generation of a signal, indicating an address violation, which is sent to the
16303	process.
16304	[EFBIG]
16305	File too large. The size of a file would exceed the maximum file size of an implementation
16306	or offset maximum established in the corresponding file description.
16307	[EHOSTUNREACH]
16308	Host is unreachable. The destination host cannot be reached (probably because the host is
16309	down or a remote router cannot reach it).
16310	[EIDRM]
16311	Identifier removed. Returned during XSI interprocess communication if an identifier has
16312	been removed from the system.
16313	[EILSEQ]
16314	Illegal byte sequence. A wide-character code has been detected that does not correspond to
16315	a valid character, or a byte sequence does not form a valid wide-character code (defined in
16316	the ISO C standard).
16317	[EINPROGRESS]
16318	Operation in progress. This code is used to indicate that an asynchronous operation has not
16319	yet completed.
16320	or:
16321	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
16322	immediately established.

16323	[EINTR]	
16324		Interrupted function call. An asynchronous signal was caught by the process during the
16325		execution of an interruptible function. If the signal handler performs a normal return, the
16326		interrupted function call may return this condition (see the Base Definitions volume of
16327		POSIX.1-2008, <signal.h>).
16328	[EINVAL]	
16329		Invalid argument. Some invalid argument was supplied; for example, specifying an
16330		undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
16331	[EIO]	
16332		Input/output error. Some physical input or output error has occurred. This error may be
16333		reported on a subsequent operation on the same file descriptor. Any other error-causing
16334		operation on the same file descriptor may cause the [EIO] error indication to be lost.
16335	[EISCONN]	
16336		Socket is connected. The specified socket is already connected.
16337	[EISDIR]	
16338		Is a directory. An attempt was made to open a directory with write mode specified.
16339	[ELOOP]	
16340		Symbolic link loop. A loop exists in symbolic links encountered during pathname
16341		resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
16342		are encountered during pathname resolution.
16343	[EMFILE]	
16344		File descriptor value too large or too many open streams. An attempt was made to open a
16345	XSI	file descriptor with a value greater than or equal to {OPEN_MAX}, or greater than or equal
16346		to the soft limit RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}); or an
16347		attempt was made to open more than the maximum number of streams allowed in the
16348		process.
16349	[EMLINK]	
16350		Too many links. An attempt was made to have the link count of a single file exceed
16351		{LINK_MAX}.
16352	[EMSGSIZE]	
16353		Message too large. A message sent on a transport provider was larger than an internal
16354		message buffer or some other network limit.
16355		or:
16356		Inappropriate message buffer length.
16357	[EMULTIHOP]	
16358		Reserved.
16359	[ENAMETOOLONG]	
16360		Filename too long. The length of a pathname exceeds {PATH_MAX} and the
16361		implementation considers this to be an error, or a pathname component is longer than
16362		{NAME_MAX}. This error may also occur when pathname substitution, as a result of
16363		encountering a symbolic link during pathname resolution, results in a pathname string the
16364		size of which exceeds {PATH_MAX}.
16365	[ENETDOWN]	
16366		Network is down. The local network interface used to reach the destination is down.

## General Information

## Error Numbers

16367		[ENETRESET]
16368		The connection was aborted by the network.
16369		[ENETUNREACH]
16370		Network unreachable. No route to the network is present.
16371		[ENFILE]
16372		Too many files open in system. Too many files are currently open in the system. The system
16373		has reached its predefined limit for simultaneously open files and temporarily cannot
16374		accept requests to open another one.
16375		[ENOBUFS]
16376		No buffer space available. Insufficient buffer resources were available in the system to
16377		perform the socket operation.
16378	OB XSR	[ENODATA]
16379		No message available. No message is available on the STREAM head read queue.
16380		[ENODEV]
16381		No such device. An attempt was made to apply an inappropriate function to a device; for
16382		example, trying to read a write-only device such as a printer.
16383		[ENOENT]
16384		No such file or directory. A component of a specified pathname does not exist, or the
16385		pathname is an empty string.
16386		[ENOEXEC]
16387		Executable file format error. A request is made to execute a file that, although it has
16388		appropriate privileges, is not in the format required by the implementation for executable
16389		files.
16390		[ENOLCK]
16391		No locks available. A system-imposed limit on the number of simultaneous file and record
16392		locks has been reached and no more are currently available.
16393		[ENOLINK]
16394		Reserved.
16395		[ENOMEM]
16396		Not enough space. The new process image requires more memory than is allowed by the
16397		hardware or system-imposed memory management constraints.
16398		[ENOMSG]
16399		No message of the desired type. The message queue does not contain a message of the
16400		required type during XSI interprocess communication.
16401		[ENOPROTOPT]
16402		Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
16403		implementation.
16404		[ENOSPC]
16405		No space left on a device. During the <i>write()</i> function on a regular file or when extending a
16406		directory, there is no free space left on the device.
16407	OB XSR	[ENOSR]
16408		No STREAM resources. Insufficient STREAMS memory resources are available to perform a
16409		STREAMS-related function. This is a temporary condition; it may be recovered from if other
16410		processes release resources.

16411	OB XSR	[ENOSTR]
16412		Not a STREAM. A STREAM function was attempted on a file descriptor that was not
16413		associated with a STREAMS device.
16414		[ENOSYS]
16415		Function not implemented. An attempt was made to use a function that is not available in
16416		this implementation.
16417		[ENOTCONN]
16418		Socket not connected. The socket is not connected.
16419		[ENOTDIR]
16420		Not a directory. A component of the specified pathname exists, but it is not a directory,
16421		when a directory was expected.
16422		[ENOTEMPTY]
16423		Directory not empty. A directory other than an empty directory was supplied when an
16424		empty directory was expected.
16425		[ENOTRECOVERABLE]
16426		State not recoverable. The state protected by a robust mutex is not recoverable.
16427		[ENOTSOCK]
16428		Not a socket. The file descriptor does not refer to a socket.
16429		[ENOTSUP]
16430		Not supported. The implementation does not support this feature of the Realtime Option
16431		Group.
16432		[ENOTTY]
16433		Inappropriate I/O control operation. A control function has been attempted for a file or
16434		special file for which the operation is inappropriate.
16435		[ENXIO]
16436		No such device or address. Input or output on a special file refers to a device that does not
16437		exist, or makes a request beyond the capabilities of the device. It may also occur when, for
16438		example, a tape drive is not on-line.
16439		[EOPNOTSUPP]
16440		Operation not supported on socket. The type of socket (address family or protocol) does not
16441		support the requested operation. A conforming implementation may assign the same values
16442		for [EOPNOTSUPP] and [ENOTSUP].
16443		[EOVERFLOW]
16444		Value too large to be stored in data type. An operation was attempted which would
16445		generate a value that is outside the range of values that can be represented in the relevant
16446		data type or that are allowed for a given data item.
16447		[EOWNERDEAD]
16448		Previous owner died. The owner of a robust mutex terminated while holding the mutex
16449		lock.
16450		[EPERM]
16451		Operation not permitted. An attempt was made to perform an operation limited to
16452		processes with appropriate privileges or to the owner of a file or other resource.
16453		[EPIPE]
16454		Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
16455		to read the data.

## General Information

## Error Numbers

16456	[EPROTO]	
16457		Protocol error. Some protocol error occurred. This error is device-specific, but is generally not related to a hardware failure.
16458		
16459	[EPROTONOSUPPORT]	
16460		Protocol not supported. The protocol is not supported by the address family, or the protocol is not supported by the implementation.
16461		
16462	[EPROTOTYPE]	
16463		Protocol wrong type for socket. The socket type is not supported by the protocol.
16464	[ERANGE]	
16465		Result too large or too small. The result of the function is too large (overflow) or too small (underflow) to be represented in the available space (defined in the ISO C standard).
16466		
16467	[EROFS]	
16468		Read-only file system. An attempt was made to modify a file or directory on a file system that is read-only.
16469		
16470	[ESPIPE]	
16471		Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
16472	[ESRCH]	
16473		No such process. No process can be found corresponding to that specified by the given process ID.
16474		
16475	[ESTALE]	
16476		Reserved.
16477	OB XSR [ETIME]	
16478		STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of this error is device-specific and could indicate either a hardware or software failure, or a timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation is unspecified.
16479		
16480		
16481		
16482	[ETIMEDOUT]	
16483		Connection timed out. The connection to a remote machine has timed out. If the connection timed out during execution of the function that reported this error (as opposed to timing out prior to the function being called), it is unspecified whether the function has completed some or all of the documented behavior associated with a successful completion of the function.
16484		
16485		
16486		
16487		
16488		or:
16489		Operation timed out. The time limit associated with the operation was exceeded before the operation completed.
16490		
16491	[EIXTBSY]	
16492		Text file busy. An attempt was made to execute a pure-procedure program that is currently open for writing, or an attempt has been made to open for writing a pure-procedure program that is being executed.
16493		
16494		
16495	[EWOULDBLOCK]	
16496		Operation would block. An operation on a socket marked as non-blocking has encountered a situation such as no data available that otherwise would have caused the function to suspend execution.
16497		
16498		
16499		A conforming implementation may assign the same values for [EWOULDBLOCK] and [EAGAIN].
16500		

16501 [EXDEV]  
 16502 Improper link. A link to a file on another file system was attempted.

### 16503 2.3.1 Additional Error Numbers

16504 Additional implementation-defined error numbers may be defined in `<errno.h>`.

## 16505 2.4 Signal Concepts

### 16506 2.4.1 Signal Generation and Delivery

16507 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes  
 16508 the signal first occurs. Examples of such events include detection of hardware faults, timer  
 16509 expiration, signals generated via the **sigevent** structure and terminal activity, as well as  
 16510 invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event  
 16511 generates signals for multiple processes.

16512 At the time of generation, a determination shall be made whether the signal has been generated  
 16513 for the process or for a specific thread within the process. Signals which are generated by some  
 16514 action attributable to a particular thread, such as a hardware fault, shall be generated for the  
 16515 thread that caused the signal to be generated. Signals that are generated in association with a  
 16516 process ID or process group ID or an asynchronous event, such as terminal activity, shall be  
 16517 generated for the process.

16518 Each process has an action to be taken in response to each signal defined by the system (see  
 16519 [Section 2.4.3](#), on page 486). A signal is said to be “delivered” to a process when the appropriate  
 16520 action for the process and signal is taken. A signal is said to be “accepted” by a process when the  
 16521 signal is selected and returned by one of the `sigwait()` functions.

16522 During the time between the generation of a signal and its delivery or acceptance, the signal is  
 16523 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a  
 16524 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal  
 16525 is anything other than to ignore the signal, and if that signal is generated for the thread, the  
 16526 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned  
 16527 by a call to the `sigwait()` function, or the action associated with it is set to ignore the signal.  
 16528 Signals generated for the process shall be delivered to exactly one of those threads within the  
 16529 process which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery  
 16530 of the signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all  
 16531 threads within the process block delivery of the signal, the signal shall remain pending on the  
 16532 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery  
 16533 of the signal, or the action associated with the signal is set to ignore the signal. If the action  
 16534 associated with a blocked signal is to ignore the signal and if that signal is generated for the  
 16535 process, it is unspecified whether the signal is discarded immediately upon generation or  
 16536 remains pending.

16537 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery  
 16538 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,  
 16539 or from the corresponding thread in the parent process if the thread was created as the result of a  
 16540 call to `fork()`. The `pthread_sigmask()`, `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control  
 16541 the manipulation of the signal mask.

16542 The determination of which action is taken in response to a signal is made at the time the signal  
 16543 is delivered, allowing for any changes since the time of generation. This determination is  
 16544 independent of the means by which the signal was originally generated. If a subsequent  
 16545 occurrence of a pending signal is generated, it is implementation-defined as to whether the  
 16546 signal is delivered or accepted more than once in circumstances other than those in which  
 16547 queuing is required. The order in which multiple, simultaneously pending signals outside the  
 16548 range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

16549 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any  
 16550 pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is  
 16551 generated for a process, all pending stop signals for that process shall be discarded. When  
 16552 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the  
 16553 SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain  
 16554 pending until it is either unblocked or a stop signal is generated for the process.

16555 An implementation shall document any condition not specified by this volume of POSIX.1-2008  
 16556 under which the implementation generates signals.

#### 16557 2.4.2 Realtime Signal Generation and Delivery

16558 This section describes functionality to support realtime signal generation and delivery.

16559 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O  
 16560 completion, interprocess message arrival, and the *sigqueue()* function, support the specification  
 16561 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**  
 16562 structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the  
 16563 following members:

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union sigval	<i>sigev_value</i>	Signal value.
void*(union sigval)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

16570 The *sigev\_notify* member specifies the notification mechanism to use when an asynchronous  
 16571 event occurs. This volume of POSIX.1-2008 defines the following values for the *sigev\_notify*  
 16572 member:

16573 SIGEV\_NONE No asynchronous notification shall be delivered when the event of  
 16574 interest occurs.

16575 SIGEV\_SIGNAL The signal specified in *sigev\_signo* shall be generated for the process when  
 16576 the event of interest occurs. If the implementation supports the Realtime  
 16577 Signals Extension option and if the SA\_SIGINFO flag is set for that signal  
 16578 number, then the signal shall be queued to the process and the value  
 16579 specified in *sigev\_value* shall be the *si\_value* component of the generated  
 16580 signal. If SA\_SIGINFO is not set for that signal number, it is unspecified  
 16581 whether the signal is queued and what value, if any, is sent.

16582 SIGEV\_THREAD A notification function shall be called to perform notification.

16583 An implementation may define additional notification mechanisms.

16584 The *sigev\_signo* member specifies the signal to be generated. The *sigev\_value* member is the  
 16585 application-defined value to be passed to the signal-catching function at the time of the signal

16586 delivery or to be returned at signal acceptance as the *si\_value* member of the **siginfo\_t** structure.

16587 The **signal** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
<b>int</b>	<i>sival_int</i>	Integer signal value.
<b>void*</b>	<i>sival_ptr</i>	Pointer signal value.

16591 The *sival\_int* member shall be used when the application-defined value is of type **int**; the  
16592 *sival\_ptr* member shall be used when the application-defined value is a pointer.

16593 When a signal is generated by the *sigqueue()* function or any signal-generating function that  
16594 supports the specification of an application-defined value, the signal shall be marked pending  
16595 and, if the SA\_SIGINFO flag is set for that signal, the signal shall be queued to the process along  
16596 with the application-specified signal value. Multiple occurrences of signals so generated are  
16597 queued in FIFO order. It is unspecified whether signals so generated are queued when the  
16598 SA\_SIGINFO flag is not set for that signal.

16599 Signals generated by the *kill()* function or other events that cause signals to occur, such as  
16600 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the  
16601 implementation does not support queuing, shall have no effect on signals already queued for the  
16602 same signal number.

16603 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the  
16604 behavior shall be as if the implementation delivers the pending unblocked signal with the  
16605 lowest signal number within that range. No other ordering of signal delivery is specified.

16606 If, when a pending signal is delivered, there are additional signals queued to that signal number,  
16607 the signal shall remain pending. Otherwise, the pending indication shall be reset.

16608 Multi-threaded programs can use an alternate event notification mechanism. When a  
16609 notification is processed, and the *sigev\_notify* member of the **sigevent** structure has the value  
16610 SIGEV\_THREAD, the function *sigev\_notify\_function* is called with parameter *sigev\_value*.

16611 The function shall be executed in an environment as if it were the *start\_routine* for a newly  
16612 created thread with thread attributes specified by *sigev\_notify\_attributes*. If *sigev\_notify\_attributes*  
16613 is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to  
16614 PTHREAD\_CREATE\_DETACHED. Supplying an attributes structure with a *detachstate* attribute  
16615 of PTHREAD\_CREATE\_JOINABLE results in undefined behavior. The signal mask of this  
16616 thread is implementation-defined.

### 16617 2.4.3 Signal Actions

16618 There are three types of action that can be associated with a signal: SIG\_DFL, SIG\_IGN, or a  
16619 pointer to a function. Initially, all signals shall be set to SIG\_DFL or SIG\_IGN prior to entry of  
16620 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows.

#### 16621 SIG\_DFL

16622 Signal-specific default action.

16623 The default actions for the signals defined in this volume of POSIX.1-2008 are specified under  
16624 **<signal.h>**. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX  
16625 shall be to terminate the process abnormally.

16626 If the default action is to terminate the process abnormally, the process is terminated as if by a  
16627 call to *\_exit()*, except that the status made available to *wait()*, *waitid()*, and *waitpid()* indicates

- 16628 XSI abnormal termination by the signal. If the default action is to terminate the process abnormally  
 16629 with additional actions, implementation-defined abnormal termination actions, such as creation  
 16630 of a core file, may also occur.
- 16631 If the default action is to stop the process, the execution of that process is temporarily  
 16632 suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process,  
 16633 unless the parent process has set the SA\_NOCLDSTOP flag. While a process is stopped, any  
 16634 additional signals that are sent to the process shall not be delivered until the process is  
 16635 continued, except SIGKILL which always terminates the receiving process. A process that is a  
 16636 member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP,  
 16637 SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a  
 16638 process, the signal shall be discarded.
- 16639 If the default action is to ignore the signal, delivery of the signal shall have no effect on the  
 16640 process.
- 16641 Setting a signal action to SIG\_DFL for a signal that is pending, and whose default action is to  
 16642 ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded,  
 16643 whether or not it is blocked. Any queued values pending shall be discarded and the resources  
 16644 used to queue them shall be released and returned to the system for other use.
- 16645 The default action for SIGCONT is to resume execution at the point where the process was  
 16646 stopped, after first handling any pending unblocked signals.
- 16647 XSI When a stopped process is continued, a SIGCHLD signal may be generated for its parent  
 16648 process, unless the parent process has set the SA\_NOCLDSTOP flag.
- 16649 **SIG\_IGN**
- 16650 Ignore signal.
- 16651 Delivery of the signal shall have no effect on the process. The behavior of a process is undefined  
 16652 after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by *kill()*,  
 16653 *sigqueue()*, or *raise()*.
- 16654 The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG\_IGN.
- 16655 Setting a signal action to SIG\_IGN for a signal that is pending shall cause the pending signal to  
 16656 be discarded, whether or not it is blocked.
- 16657 If a process sets the action for the SIGCHLD signal to SIG\_IGN, the behavior is unspecified,  
 16658 XSI except as specified below.
- 16659 If the action for the SIGCHLD signal is set to SIG\_IGN, child processes of the calling processes  
 16660 shall not be transformed into zombie processes when they terminate. If the calling process  
 16661 subsequently waits for its children, and the process has no unwaited-for children that were  
 16662 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,  
 16663 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].
- 16664 Any queued values pending shall be discarded and the resources used to queue them shall be  
 16665 released and made available to queue other signals.

16666 **Pointer to a Function**

16667 Catch signal.

16668 On delivery of the signal, the receiving process is to execute the signal-catching function at the  
 16669 specified address. After returning from the signal-catching function, the receiving process shall  
 16670 resume execution at the point at which it was interrupted.

16671 If the SA\_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as  
 16672 a C-language function call as follows:

```
16673 void func(int signo);
```

16674 If the SA\_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-  
 16675 language function call as follows:

```
16676 void func(int signo, siginfo_t *info, void *context);
```

16677 where *func* is the specified signal-catching function, *signo* is the signal number of the signal  
 16678 being delivered, and *info* is a pointer to a **siginfo\_t** structure defined in **<signal.h>** containing at  
 16679 least the following members:

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
pid_t	<i>si_pid</i>	Sending process ID.
uid_t	<i>si_uid</i>	Real user ID of sending process.
void *	<i>si_addr</i>	Address of faulting instruction.
int	<i>si_status</i>	Exit value or signal.
union signal	<i>si_value</i>	Signal value.

16688 The *si\_signo* member shall contain the signal number. This shall be the same as the *signo*  
 16689 parameter. The *si\_code* member shall contain a code identifying the cause of the signal. The  
 16690 following non-signal-specific values are defined for *si\_code*:

16691 SI\_USER The signal was sent by the *kill()* function. The implementation may set *si\_code*  
 16692 to SI\_USER if the signal was sent by the *raise()* or *abort()* functions or any  
 16693 similar functions provided as implementation extensions.

16694 SI\_QUEUE The signal was sent by the *sigqueue()* function.

16695 SI\_TIMER The signal was generated by the expiration of a timer set by *timer\_settime()*.

16696 SI\_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

16697 MSG SI\_MESGQ The signal was generated by the arrival of a message on an empty message  
 16698 queue.

16699 Signal-specific values for *si\_code* are also defined, as described in XBD **<signal.h>**.

16700 If the signal was not generated by one of the functions or events listed above, *si\_code* shall be set  
 16701 either to one of the signal-specific values described in XBD **<signal.h>**, or to an implementation-  
 16702 defined value that is not equal to any of the values defined above.

16703 XSI If *si\_code* is SI\_USER or SI\_QUEUE, or any value less than or equal to 0, then the signal was  
 16704 generated by a process and *si\_pid* and *si\_uid* shall be set to the process ID and the real user ID of  
 16705 the sender, respectively.

16706 In addition, *si\_addr*, *si\_pid*, *si\_status*, and *si\_uid* shall be set for certain signal-specific values of  
 16707 *si\_code*, as described in XBD **<signal.h>**.

16708 If *si\_code* is one of SI\_QUEUE, SI\_TIMER, SI\_ASYNCIO, or SI\_MESGQ, then *si\_value* shall

## General Information

## Signal Concepts

- 16709 contain the application-specified signal value. Otherwise, the contents of *si\_value* are undefined.
- 16710 The behavior of a process is undefined after it returns normally from a signal-catching function  
16711 for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill()*, *sigqueue()*,  
16712 or *raise()*.
- 16713 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.
- 16714 If a process establishes a signal-catching function for the SIGCHLD signal while it has a  
16715 terminated child process for which it has not waited, it is unspecified whether a SIGCHLD  
16716 signal is generated to indicate that child process.
- 16717 When signal-catching functions are invoked asynchronously with process execution, the  
16718 behavior of some of the functions defined by this volume of POSIX.1-2008 is unspecified if they  
16719 are called from a signal-catching function.
- 16720 The following table defines a set of functions that shall be async-signal-safe. Therefore,  
16721 applications can invoke them, without restriction, from signal-catching functions:

16722	<i>_Exit()</i>	<i>fexecve()</i>	<i>poll()</i>	<i>sigqueue()</i>
16723	<i>_exit()</i>	<i>fork()</i>	<i>posix_trace_event()</i>	<i>sigset()</i>
16724	<i>abort()</i>	<i>fstat()</i>	<i>pselect()</i>	<i>sigsuspend()</i>
16725	<i>accept()</i>	<i>fstatat()</i>	<i>raise()</i>	<i>sleep()</i>
16726	<i>access()</i>	<i>fsync()</i>	<i>read()</i>	<i>socketmark()</i>
16727	<i>aio_error()</i>	<i>ftruncate()</i>	<i>readlink()</i>	<i>socket()</i>
16728	<i>aio_return()</i>	<i>futimens()</i>	<i>readlinkat()</i>	<i>socketpair()</i>
16729	<i>aio_suspend()</i>	<i>getegid()</i>	<i>recv()</i>	<i>stat()</i>
16730	<i>alarm()</i>	<i>geteuid()</i>	<i>recvfrom()</i>	<i>symlink()</i>
16731	<i>bind()</i>	<i>getgid()</i>	<i>recvmsg()</i>	<i>symlinkat()</i>
16732	<i>cfgetispeed()</i>	<i>getgroups()</i>	<i>rename()</i>	<i>tcdrain()</i>
16733	<i>cfgetospeed()</i>	<i>getpeername()</i>	<i>renameat()</i>	<i>tcflow()</i>
16734	<i>cfsetispeed()</i>	<i>getpgrp()</i>	<i>rmdir()</i>	<i>tcflush()</i>
16735	<i>cfsetospeed()</i>	<i>getpid()</i>	<i>select()</i>	<i>tcgetattr()</i>
16736	<i>chdir()</i>	<i>getppid()</i>	<i>sem_post()</i>	<i>tcgetpgrp()</i>
16737	<i>chmod()</i>	<i>getsockname()</i>	<i>send()</i>	<i>tcsendbreak()</i>
16738	<i>chown()</i>	<i>getsockopt()</i>	<i>sendmsg()</i>	<i>tcsetattr()</i>
16739	<i>clock_gettime()</i>	<i>getuid()</i>	<i>sendto()</i>	<i>tcsetpgrp()</i>
16740	<i>close()</i>	<i>kill()</i>	<i>setgid()</i>	<i>time()</i>
16741	<i>connect()</i>	<i>link()</i>	<i>setpgid()</i>	<i>timer_getoverrun()</i>
16742	<i>creat()</i>	<i>linkat()</i>	<i>setsid()</i>	<i>timer_gettime()</i>
16743	<i>dup()</i>	<i>listen()</i>	<i>setsockopt()</i>	<i>timer_settime()</i>
16744	<i>dup2()</i>	<i>lseek()</i>	<i>setuid()</i>	<i>times()</i>
16745	<i>execl()</i>	<i>lstat()</i>	<i>shutdown()</i>	<i>umask()</i>
16746	<i>execle()</i>	<i>mkdir()</i>	<i>sigaction()</i>	<i>uname()</i>
16747	<i>execv()</i>	<i>mkdirat()</i>	<i>sigaddset()</i>	<i>unlink()</i>
16748	<i>execve()</i>	<i>mkfifo()</i>	<i>sigdelset()</i>	<i>unlinkat()</i>
16749	<i>faccessat()</i>	<i>mkfifoat()</i>	<i>sigemptyset()</i>	<i>utime()</i>
16750	<i>fchmod()</i>	<i>mknod()</i>	<i>sigfillset()</i>	<i>utimensat()</i>
16751	<i>fchmodat()</i>	<i>mknodat()</i>	<i>sigismember()</i>	<i>utimes()</i>
16752	<i>fchown()</i>	<i>open()</i>	<i>signal()</i>	<i>wait()</i>
16753	<i>fchownat()</i>	<i>openat()</i>	<i>sigpause()</i>	<i>waitpid()</i>
16754	<i>fcntl()</i>	<i>pause()</i>	<i>sigpending()</i>	<i>write()</i>
16755	<i>fdatasync()</i>	<i>pipe()</i>	<i>sigprocmask()</i>	

- 16756 All functions not in the above table are considered to be unsafe with respect to signals. In the  
16757 presence of signals, all functions defined by this volume of POSIX.1-2008 shall behave as defined

16758 when called from or interrupted by a signal-catching function, with a single exception: when a  
 16759 signal interrupts an unsafe function and the signal-catching function calls an unsafe function,  
 16760 the behavior is undefined.

16761 Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be  
 16762 *async-signal-safe*.

16763 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or  
 16764 continue, the entire process shall be terminated, stopped, or continued, respectively.

#### 16765 2.4.4 Signal Effects on Other Functions

16766 Signals affect the behavior of certain functions defined by this volume of POSIX.1-2008 if  
 16767 delivered to a process while it is executing such a function. If the action of the signal is to  
 16768 terminate the process, the process shall be terminated and the function shall not return. If the  
 16769 action of the signal is to stop the process, the process shall stop until continued or terminated.  
 16770 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the  
 16771 original function shall continue at the point the process was stopped. If the action of the signal is  
 16772 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case  
 16773 the original function is said to be “interrupted” by the signal. If the signal-catching function  
 16774 executes a **return** statement, the behavior of the interrupted function shall be as described  
 16775 individually for that function, except as noted for unsafe functions. Signals that are ignored shall  
 16776 not affect the behavior of any function; signals that are blocked shall not affect the behavior of  
 16777 any function until they are unblocked and then delivered, except as specified for the *sigpending()*  
 16778 and *sigwait()* functions.

### 16779 2.5 Standard I/O Streams

16780 CX A stream is associated with an external file (which may be a physical device) or memory buffer  
 16781 CX by “opening” a file or buffer. This may involve “creating” a new file. Creating an existing file  
 16782 causes its former contents to be discarded if necessary. If a file can support positioning requests  
 16783 (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the  
 16784 stream is positioned at the start (byte number 0) of the file, unless the file is opened with append  
 16785 mode, in which case it is implementation-defined whether the file position indicator is initially  
 16786 positioned at the beginning or end of the file. The file position indicator is maintained by  
 16787 subsequent reads, writes, and positioning requests, to facilitate an orderly progression through  
 16788 the file. All input takes place as if bytes were read by successive calls to *fgetc()*; all output takes  
 16789 place as if bytes were written by successive calls to *fputc()*.

16790 When a stream is “unbuffered”, bytes are intended to appear from the source or at the  
 16791 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a  
 16792 block. When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a  
 16793 buffer is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block  
 16794 when a <newline> byte is encountered. Furthermore, bytes are intended to be transmitted as a  
 16795 block when a buffer is filled, when input is requested on an unbuffered stream, or when input is  
 16796 requested on a line-buffered stream that requires the transmission of bytes. Support for these  
 16797 characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

16798 A file may be disassociated from a controlling stream by “closing” the file. Output streams are  
 16799 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from  
 16800 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed  
 16801 (including the standard streams).

16802 A file may be subsequently reopened, by the same or another program execution, and its  
 16803 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function  
 16804 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all  
 16805 output streams are flushed) before program termination. Other paths to program termination,  
 16806 such as calling *abort()*, need not close all files properly.

16807 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**  
 16808 object need not necessarily serve in place of the original.

16809 At program start-up, three streams are predefined and need not be opened explicitly: *standard*  
 16810 *input* (for reading conventional input), *standard output* (for writing conventional output), and  
 16811 *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 16812 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 16813 the stream can be determined not to refer to an interactive device.

16814 CX A stream associated with a memory buffer shall have the same operations for text files that a  
 16815 stream associated with an external file would have. In addition, the stream orientation shall be  
 16816 determined in exactly the same fashion.

16817 Input and output operations on a stream associated with a memory buffer by a call to  
 16818 *fmemopen()* shall be constrained by the implementation to take place within the bounds of the  
 16819 memory buffer. In the case of a stream opened by *open\_memstream()* or *open\_wmemstream()*, the  
 16820 memory area shall grow dynamically to accommodate write operations as necessary. For output,  
 16821 data is moved from the buffer provided by *setvbuf()* to the memory stream during a flush or  
 16822 close operation.

## 16823 2.5.1 Interaction of File Descriptors and Standard I/O Streams

16824 CX This section describes the interaction of file descriptors and standard I/O streams. The  
 16825 functionality described in this section is an extension to the ISO C standard (and the rest of this  
 16826 section is not further CX shaded).

16827 An open file description may be accessed through a file descriptor, which is created using  
 16828 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as  
 16829 *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file  
 16830 description to which it refers; an open file description may have several handles.

16831 Handles can be created or destroyed by explicit user action, without affecting the underlying  
 16832 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,  
 16833 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

16834 A file descriptor that is never used in an operation that could affect the file offset (for example,  
 16835 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to  
 16836 one (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include  
 16837 the file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is  
 16838 not used directly by the application to affect the file offset. The *read()* and *write()* functions  
 16839 implicitly affect the file offset; *lseek()* explicitly affects it.

16840 The result of function calls involving any one handle (the “active handle”) is defined elsewhere  
 16841 in this volume of POSIX.1-2008, but if two or more handles are used, and any one of them is a  
 16842 stream, the application shall ensure that their actions are coordinated as described below. If this  
 16843 is not done, the result is undefined.

16844 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is  
 16845 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same  
 16846 open file description as its previous value), or when the process owning that stream terminates  
 16847 with *exit()*, *abort()*, or due to a signal. A file descriptor is closed by *close()*, *\_exit()*, or the *exec*

- 16848 functions when `FD_CLOEXEC` is set on that file descriptor.
- 16849 For a handle to become the active handle, the application shall ensure that the actions below are  
 16850 performed between the last use of the handle (the current active handle) and the first use of the  
 16851 second handle (the future active handle). The second handle then becomes the active handle. All  
 16852 activity by the application affecting the file offset on the first handle shall be suspended until it  
 16853 again becomes the active file handle. (If a stream function has as an underlying function one that  
 16854 affects the file offset, the stream function shall be considered to affect the file offset.)
- 16855 The handles need not be in the same process for these rules to apply.
- 16856 Note that after a `fork()`, two handles exist where one existed before. The application shall ensure  
 16857 that, if both handles can ever be accessed, they are both in a state where the other could become  
 16858 the active handle first. The application shall prepare for a `fork()` exactly as if it were a change of  
 16859 active handle. (If the only action performed by one of the processes is one of the `exec` functions or  
 16860 `_exit()` (not `exit()`), the handle is never accessed in that process.)
- 16861 For the first handle, the first applicable condition below applies. After the actions required  
 16862 below are taken, if the handle is still open, the application can close it.
- 16863 • If it is a file descriptor, no action is required.
  - 16864 • If the only further action to be performed on any handle to this open file descriptor is to  
 16865 close it, no action need be taken.
  - 16866 • If it is a stream which is unbuffered, no action need be taken.
  - 16867 • If it is a stream which is line buffered, and the last byte written to the stream was a  
 16868 <newline> (that is, as if a:
- ```
16869         putchar('\n')
```
- 16870 was the most recent operation on that stream), no action need be taken.
- 16871 • If it is a stream which is open for writing or appending (but not also open for reading), the  
 16872 application shall either perform an `fflush()`, or the stream shall be closed.
  - 16873 • If the stream is open for reading and it is at the end of the file (`feof()` is true), no action need  
 16874 be taken.
  - 16875 • If the stream is open with a mode that allows reading and the underlying open file  
 16876 description refers to a device that is capable of seeking, the application shall either perform  
 16877 an `fflush()`, or the stream shall be closed.
- 16878 Otherwise, the result is undefined.
- 16879 For the second handle:
- 16880 • If any previous active handle has been used by a function that explicitly changed the file  
 16881 offset, except as required above for the first handle, the application shall perform an `lseek()`  
 16882 or `fseek()` (as appropriate to the type of handle) to an appropriate location.
- 16883 If the active handle ceases to be accessible before the requirements on the first handle, above,  
 16884 have been met, the state of the open file description becomes undefined. This might occur  
 16885 during functions such as a `fork()` or `_exit()`.
- 16886 The `exec` functions make inaccessible all streams that are open at the time they are called,  
 16887 independent of which streams or file descriptors may be available to the new process image.
- 16888 When these rules are followed, regardless of the sequence of handles used, implementations  
 16889 shall ensure that an application, even one consisting of several processes, shall yield correct

16890 results: no data shall be lost or duplicated when writing, and all data shall be written in order,  
 16891 except as requested by seeks. It is implementation-defined whether, and under what conditions,  
 16892 all input is seen exactly once.

16893 If the rules above are not followed, the result is unspecified.

16894 Each function that operates on a stream is said to have zero or more “underlying functions”.  
 16895 This means that the stream function shares certain traits with the underlying functions, but does  
 16896 not require that there be any relation between the implementations of the stream function and its  
 16897 underlying functions.

## 16898 2.5.2 Stream Orientation and Encoding Rules

16899 For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an  
 16900 “orientation”. After a stream is associated with an external file, but before any operations are  
 16901 performed on it, the stream is without orientation. Once a wide-character input/output function  
 16902 has been applied to a stream without orientation, the stream shall become “wide-oriented”.  
 16903 Similarly, once a byte input/output function has been applied to a stream without orientation,  
 16904 the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()*  
 16905 function can otherwise alter the orientation of a stream.

16906 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*  
 16907 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

16908 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character  
 16909 input/output functions cannot be applied to a byte-oriented stream. The remaining stream  
 16910 operations shall not affect and shall not be affected by a stream’s orientation, except for the  
 16911 following additional restriction:

- 16912 • For wide-oriented streams, after a successful call to a file-positioning function that leaves  
 16913 the file position indicator prior to the end-of-file, a wide-character output function can  
 16914 overwrite a partial character; any file contents beyond the byte(s) written are henceforth  
 16915 undefined.

16916 Each wide-oriented stream has an associated **mbstate\_t** object that stores the current parse state  
 16917 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this  
 16918 **mbstate\_t** object as part of the value of the **fpos\_t** object. A later successful call to *fsetpos()* using  
 16919 the same stored **fpos\_t** value shall restore the value of the associated **mbstate\_t** object as well as  
 16920 the position within the controlled stream.

16921 Implementations that support multiple encoding rules associate an encoding rule with the  
 16922 stream. The encoding rule shall be determined by the setting of the *LC\_CTYPE* category in the  
 16923 current locale at the time when the stream becomes wide-oriented. As with the stream’s  
 16924 orientation, the encoding rule associated with a stream cannot be changed once it has been set,  
 16925 except by a successful call to *freopen()* which clears the encoding rule and resets the orientation  
 16926 to unoriented.

16927 Although wide-oriented streams are conceptually sequences of wide characters, the external file  
 16928 associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters  
 16929 generalized as follows:

- 16930 • Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte  
 16931 encodings valid for use internal to the program).
- 16932 • A file need not begin nor end in the initial shift state.

16933 Moreover, the encodings used for characters may differ among files. Both the nature and choice  
 16934 of such encodings are implementation-defined.

16935 The wide-character input functions read characters from the stream and convert them to wide  
 16936 characters as if they were read by successive calls to the *fgetc()* function. Each conversion shall  
 16937 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the  
 16938 CX stream's own **mbstate\_t** object, except the encoding rule associated with the stream is used  
 16939 instead of the encoding rule implied by the *LC\_CTYPE* category of the current locale.

16940 The wide-character output functions convert wide characters to (possibly multi-byte) characters  
 16941 and write them to the stream as if they were written by successive calls to the *fputc()* function.  
 16942 Each conversion shall occur as if by a call to the *wctomb()* function, with the conversion state  
 16943 CX described by the stream's own **mbstate\_t** object, except the encoding rule associated with the  
 16944 stream is used instead of the encoding rule implied by the *LC\_CTYPE* category of the current  
 16945 locale.

16946 An "encoding error" shall occur if the character sequence presented to the underlying *mbrtowc()*  
 16947 function does not form a valid (generalized) character, or if the code value passed to the  
 16948 underlying *wctomb()* function does not correspond to a valid (generalized) character. The wide-  
 16949 character input/output functions and the byte input/output functions store the value of the  
 16950 macro [EILSEQ] in *errno* if and only if an encoding error occurs.

## 16951 2.6 STREAMS

16952 OB XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option  
 16953 Group. The functionality described in this section is dependent on support of the XSI STREAMS  
 16954 option (and the rest of this section is not further shaded for this option).

16955 STREAMS provides a uniform mechanism for implementing networking services and other  
 16956 character-based I/O. The STREAMS function provides direct access to protocol modules.  
 16957 STREAMS modules are unspecified objects. Access to STREAMS modules is provided by  
 16958 interfaces in POSIX.1-2008. Creation of STREAMS modules is outside the scope of  
 16959 POSIX.1-2008.

16960 A STREAM is typically a full-duplex connection between a process and an open device or  
 16961 pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex  
 16962 connection between two processes. The STREAM itself exists entirely within the implementation  
 16963 and provides a general character I/O function for processes. It optionally includes one or more  
 16964 intermediate processing modules that are interposed between the process end of the STREAM  
 16965 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

16966 STREAMS I/O is based on messages. There are three types of message:

- 16967 • *Data messages* containing actual data for input or output
- 16968 • *Control data* containing instructions for the STREAMS modules and underlying  
 16969 implementation
- 16970 • Other messages, which include file descriptors

16971 The interface between the STREAM and the rest of the implementation is provided by a set of  
 16972 functions at the STREAM head. When a process calls *write()*, *writev()*, *putmsg()*, *putpmsg()*, or  
 16973 *ioctl()*, messages are sent down the STREAM, and *read()*, *readv()*, *getmsg()*, or *getpmsg()* accepts  
 16974 data from the STREAM and passes it to a process. Data intended for the device at the  
 16975 downstream end of the STREAM is packaged into messages and sent downstream, while data  
 16976 and signals from the device are composed into messages by the device driver and sent upstream  
 16977 to the STREAM head.

16978 When a STREAMS-based device is opened, a STREAM shall be created that contains the

16979 STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an  
 16980 implementation, when a pipe is created, two STREAMS shall be created, each containing a  
 16981 STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are  
 16982 “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the  
 16983 STREAM was a push-down stack.

#### 16984 **Priority**

16985 Message types are classified according to their queuing priority and may be *normal* (non-  
 16986 priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that  
 16987 determines its ordering when placed on a queue. Normal messages have a priority band of 0  
 16988 and shall always be placed at the end of the queue following all other messages in the queue.  
 16989 High-priority messages are always placed at the head of a queue, but shall be discarded if there  
 16990 is already a high-priority message in the queue. Their priority band shall be ignored; they are  
 16991 high-priority by virtue of their type. Priority messages have a priority band greater than 0.  
 16992 Priority messages are always placed after any messages of the same or higher priority. High-  
 16993 priority and priority messages are used to send control and data information outside the normal  
 16994 flow of control. By convention, high-priority messages shall not be affected by flow control.  
 16995 Normal and priority messages have separate flow controls.

#### 16996 **Message Parts**

16997 A process may access STREAMS messages that contain a data part, control part, or both. The  
 16998 data part is that information which is transmitted over the communication medium and the  
 16999 control information is used by the local STREAMS modules. The other types of messages are  
 17000 used between modules and are not accessible to processes. Messages containing only a data part  
 17001 are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writv()*.  
 17002 Messages containing a control part with or without a data part are accessible via calls to  
 17003 *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

### 17004 **2.6.1 Accessing STREAMS**

17005 A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*,  
 17006 *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable  
 17007 function definitions for general properties and errors.

17008 Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor  
 17009 *fildevs*. The control functions may be performed by the STREAM head, a STREAMS module, or  
 17010 the STREAMS driver for the STREAM.

17011 STREAMS modules and drivers can detect errors, sending an error message to the STREAM  
 17012 head, thus causing subsequent functions to fail and set *errno* to the value specified in the  
 17013 message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request  
 17014 alone by sending a negative acknowledgement message to the STREAM head. This shall cause  
 17015 just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

17016 **2.7 XSI Interprocess Communication**

17017 XSI This section describes extensions to support interprocess communication. The functionality  
 17018 described in this section shall be provided on implementations that support the XSI option (and  
 17019 the rest of this section is not further shaded).

17020 The following message passing, semaphore, and shared memory services form an XSI  
 17021 interprocess communication facility. Certain aspects of their operation are common, and are  
 17022 defined as follows.

| IPC Functions   |                 |                 |
|-----------------|-----------------|-----------------|
| <i>msgctl()</i> | <i>semctl()</i> | <i>shmat()</i>  |
| <i>msgget()</i> | <i>semget()</i> | <i>shmctl()</i> |
| <i>msgrcv()</i> | <i>semop()</i>  | <i>shmdt()</i>  |
| <i>msgsnd()</i> |                 | <i>shmget()</i> |

17028 Another interprocess communication facility is provided by functions in the Realtime Option  
 17029 Group; see Section 2.8 (on page 497).

17030 **2.7.1 IPC General Description**

17031 Each individual shared memory segment, message queue, and semaphore set shall be identified  
 17032 by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore  
 17033 identifier, *semid*, and a message queue identifier, *msgid*. The identifiers shall be returned by calls  
 17034 to *shmget()*, *semget()*, and *msgget()*, respectively.

17035 Associated with each identifier is a data structure which contains data related to the operations  
 17036 which may be or may have been performed; see the Base Definitions volume of POSIX.1-2008,  
 17037 <**sys/shm.h**>, <**sys/sem.h**>, and <**sys/msg.h**> for their descriptions.

17038 Each of the data structures contains both ownership information and an **ipc\_perm** structure (see  
 17039 the Base Definitions volume of POSIX.1-2008, <**sys/ipc.h**>) which are used in conjunction to  
 17040 determine whether or not read/write (read/alter for semaphores) permissions should be  
 17041 granted to processes using the IPC facilities. The *mode* member of the **ipc\_perm** structure acts as  
 17042 a bit field which determines the permissions.

17043 The values of the bits are given below in octal notation.

| Bit  | Meaning          |
|------|------------------|
| 0400 | Read by user.    |
| 0200 | Write by user.   |
| 0040 | Read by group.   |
| 0020 | Write by group.  |
| 0004 | Read by others.  |
| 0002 | Write by others. |

17051 The name of the **ipc\_perm** structure is *shm\_perm*, *sem\_perm*, or *msg\_perm*, depending on which  
 17052 service is being used. In each case, read and write/alter permissions shall be granted to a  
 17053 process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as  
 17054 appropriate):

- 17055 • The process has appropriate privileges.
- 17056 • The effective user ID of the process matches *xxx\_perm.cuid* or *xxx\_perm.uid* in the data  
 17057 structure associated with the IPC identifier, and the appropriate bit of the *user* field in  
 17058 *xxx\_perm.mode* is set.

- 17059 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* but the  
 17060 effective group ID of the process matches *xxx\_perm.cgid* or *xxx\_perm.gid* in the data  
 17061 structure associated with the IPC identifier, and the appropriate bit of the *group* field in  
 17062 *xxx\_perm.mode* is set.
- 17063 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* and the  
 17064 effective group ID of the process does not match *xxx\_perm.cgid* or *xxx\_perm.gid* in the data  
 17065 structure associated with the IPC identifier, but the appropriate bit of the *other* field in  
 17066 *xxx\_perm.mode* is set.
- 17067 Otherwise, the permission shall be denied.

## 17068 2.8 Realtime

17069 This section defines functions to support the source portability of applications with realtime  
 17070 requirements. The presence of some of these functions is dependent on support for  
 17071 implementation options described in the text.

17072 The specific functional areas included in this section and their scope include the following. Full  
 17073 definitions of these terms can be found in XBD Chapter 3 (on page 33).

- 17074 • Semaphores
- 17075 • Process Memory Locking
- 17076 • Memory Mapped Files and Shared Memory Objects
- 17077 • Priority Scheduling
- 17078 • Realtime Signal Extension
- 17079 • Timers
- 17080 • Interprocess Communication
- 17081 • Synchronized Input and Output
- 17082 • Asynchronous Input and Output

17083 All the realtime functions defined in this volume of POSIX.1-2008 are portable, although some of  
 17084 the numeric parameters used by an implementation may have hardware dependencies.

### 17085 2.8.1 Realtime Signals

17086 See Section 2.4.2 (on page 485).

### 17087 2.8.2 Asynchronous I/O

17088 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O  
 17089 functions. It is defined in the Base Definitions volume of POSIX.1-2008, <**aiocb.h**> and has at least  
 17090 the following members:

|       | Member Type            | Member Name           | Description                |
|-------|------------------------|-----------------------|----------------------------|
| 17091 | <b>int</b>             | <i>aio_fildes</i>     | File descriptor.           |
| 17092 | <b>off_t</b>           | <i>aio_offset</i>     | File offset.               |
| 17093 | <b>volatile void*</b>  | <i>aio_buf</i>        | Location of buffer.        |
| 17094 | <b>size_t</b>          | <i>aio_nbytes</i>     | Length of transfer.        |
| 17095 | <b>int</b>             | <i>aio_reqprio</i>    | Request priority offset.   |
| 17096 | <b>struct sigevent</b> | <i>aio_sigevent</i>   | Signal number and value.   |
| 17097 | <b>int</b>             | <i>aio_lio_opcode</i> | Operation to be performed. |
| 17098 |                        |                       |                            |

17099 The *aio\_fildes* element is the file descriptor on which the asynchronous operation is performed.

17100 If O\_APPEND is not set for the file descriptor *aio\_fildes* and if *aio\_fildes* is associated with a  
 17101 device that is capable of seeking, then the requested operation takes place at the absolute  
 17102 position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the  
 17103 operation with an *offset* argument equal to *aio\_offset* and a *whence* argument equal to SEEK\_SET.  
 17104 If O\_APPEND is set for the file descriptor, or if *aio\_fildes* is associated with a device that is  
 17105 incapable of seeking, write operations append to the file in the same order as the calls were  
 17106 made, with the following exception: under implementation-defined circumstances, such as  
 17107 operation on a multi-processor or when requests of differing priorities are submitted at the same  
 17108 time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming  
 17109 application to determine whether this relaxation applies, all strictly conforming applications  
 17110 which rely on ordering of output shall be written in such a way that they will operate correctly if  
 17111 the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the  
 17112 value of the file offset for the file is unspecified. The *aio\_nbytes* and *aio\_buf* elements are the same  
 17113 as the *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

17114 If \_POSIX\_PRIORITIZED\_IO and \_POSIX\_PRIORITY\_SCHEDULING are defined, then  
 17115 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation  
 17116 based on the current scheduling priority of the calling process. The *aio\_reqprio* member can be  
 17117 used to lower (but not raise) the asynchronous I/O operation priority and is within the range  
 17118 zero through {AIO\_PRIO\_DELTA\_MAX}, inclusive. Unless both \_POSIX\_PRIORITIZED\_IO and  
 17119 \_POSIX\_PRIORITY\_SCHEDULING are defined, the order of processing asynchronous I/O  
 17120 requests is unspecified. When both \_POSIX\_PRIORITIZED\_IO and  
 17121 \_POSIX\_PRIORITY\_SCHEDULING are defined, the order of processing of requests submitted  
 17122 by processes whose schedulers are not SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC is  
 17123 unspecified. The priority of an asynchronous request is computed as (process scheduling  
 17124 priority) minus *aio\_reqprio*. The priority assigned to each asynchronous I/O request is an  
 17125 indication of the desired order of execution of the request relative to other asynchronous I/O  
 17126 requests for this file. If \_POSIX\_PRIORITIZED\_IO is defined, requests issued with the same  
 17127 priority to a character special file are processed by the underlying device in FIFO order; the  
 17128 order of processing of requests of the same priority issued to files that are not character special  
 17129 files is unspecified. Numerically higher priority values indicate requests of higher priority. The  
 17130 value of *aio\_reqprio* has no effect on process scheduling priority. When prioritized asynchronous  
 17131 I/O requests to the same file are blocked waiting for a resource required for that I/O operation,  
 17132 the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests  
 17133 are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is  
 17134 implementation-defined. If \_POSIX\_PRIORITIZED\_IO is defined, the implementation shall  
 17135 define for which files I/O prioritization is supported.

17136 The *aio\_sigevent* determines how the calling process shall be notified upon I/O completion, as  
 17137 specified in Section 2.4.1 (on page 484). If *aio\_sigevent.sigev\_notify* is SIGEV\_NONE, then no  
 17138 signal shall be posted upon I/O completion, but the error status for the operation and the return  
 17139 status for the operation shall be set appropriately.

17140 The *aio\_lio\_opcode* field is used only by the *lio\_listio()* call. The *lio\_listio()* call allows multiple

17141 asynchronous I/O operations to be submitted at a single time. The function takes as an  
 17142 argument an array of pointers to **aio** structures. Each **aio** structure indicates the operation to  
 17143 be performed (read or write) via the *aio\_lio\_opcode* field.

17144 The address of the **aio** structure is used as a handle for retrieving the error status and return  
 17145 status of the asynchronous operation while it is in progress.

17146 The **aio** structure and the data buffers associated with the asynchronous I/O operation are  
 17147 being used by the system for asynchronous I/O while, and only while, the error status of the  
 17148 asynchronous operation is equal to [EINPROGRESS]. Applications shall not modify the **aio**  
 17149 structure while the structure is being used by the system for asynchronous I/O.

17150 The return status of the asynchronous operation is the number of bytes transferred by the I/O  
 17151 operation. If the error status is set to indicate an error completion, then the return status is set to  
 17152 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.  
 17153 When the error status is not equal to [EINPROGRESS], the return status shall reflect the return  
 17154 status of the corresponding synchronous operation.

### 17155 2.8.3 Memory Management

#### 17156 2.8.3.1 Memory Locking

17157 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict  
 17158 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,  
 17159 is the value of the configurable system variable {PAGESIZE}. If an implementation has no  
 17160 restrictions on size or alignment, it may specify a 1-byte page size.

17161 ML|MLR Memory locking guarantees the residence of portions of the address space. It is implementation-  
 17162 defined whether locking memory guarantees fixed translation between virtual addresses (as  
 17163 seen by the process) and physical addresses. Per-process memory locks are not inherited across a  
 17164 *fork()*, and all memory locks owned by a process are unlocked upon *exec* or process termination.  
 17165 Unmapping of an address range removes any memory locks established on that address range  
 17166 by this process.

#### 17167 2.8.3.2 Memory Mapped Files

17168 Range memory mapping operations are defined in terms of pages. Implementations may  
 17169 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,  
 17170 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has  
 17171 no restrictions on size or alignment, it may specify a 1-byte page size.

17172 Memory mapped files provide a mechanism that allows a process to access files by directly  
 17173 incorporating file data into its address space. Once a file is mapped into a process address space,  
 17174 the data can be manipulated as memory. If more than one process maps a file, its contents are  
 17175 shared among them. If the mappings allow shared write access, then data written into the  
 17176 memory object through the address space of one process appears in the address spaces of all  
 17177 processes that similarly map the same portion of the memory object.

17178 SHM Shared memory objects are named regions of storage that may be independent of the file system  
 17179 and can be mapped into the address space of one or more processes to allow them to share the  
 17180 associated memory.

17181 SHM An *unlink()* of a file or *shm\_unlink()* of a shared memory object, while causing the removal of  
 17182 the name, does not unmap any mappings established for the object. Once the name has been  
 17183 removed, the contents of the memory object are preserved as long as it is referenced. The  
 17184 memory object remains referenced as long as a process has the memory object open or has some  
 17185 area of the memory object mapped.

#### 17186 2.8.3.3 Memory Protection

17187 When an object is mapped, various application accesses to the mapped region may result in  
 17188 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and  
 17189 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 17190 • A mapping may be restricted to disallow some types of access.
- 17191 • Write attempts to memory that was mapped without write access, or any access to  
 17192 memory mapped PROT\_NONE, shall result in a SIGSEGV signal.
- 17193 • References to unmapped addresses shall result in a SIGSEGV signal.
- 17194 • Reference to whole pages within the mapping, but beyond the current length of the object,  
 17195 shall result in a SIGBUS signal.
- 17196 • The size of the object is unaffected by access beyond the end of the object (even if a  
 17197 SIGBUS is not generated).

#### 17198 2.8.3.4 Typed Memory Objects

17199 TYM The functionality described in this section shall be provided on implementations that support  
 17200 the Typed Memory Objects option (and the rest of this section is not further shaded for this  
 17201 option).

17202 Implementations may support the Typed Memory Objects option independently of support for  
 17203 memory mapped files or shared memory objects. Typed memory objects are implementation-  
 17204 configurable named storage pools accessible from one or more processors in a system, each via  
 17205 one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid  
 17206 combination of a storage pool and a port is identified through a name that is defined at system  
 17207 configuration time, in an implementation-defined manner; the name may be independent of the  
 17208 file system. Using this name, a typed memory object can be opened and mapped into process  
 17209 address space. For a given storage pool and port, it is necessary to support both dynamic  
 17210 allocation from the pool as well as mapping at an application-supplied offset within the pool;  
 17211 when dynamic allocation has been performed, subsequent deallocation must be supported.  
 17212 Lastly, accessing typed memory objects from different ports requires a method for obtaining the  
 17213 offset and length of contiguous storage of a region of typed memory (dynamically allocated or  
 17214 not); this allows typed memory to be shared among processes and/or processors while being  
 17215 accessed from the desired port.

17216 **2.8.4 Process Scheduling**

17217 PS The functionality described in this section shall be provided on implementations that support  
 17218 the Process Scheduling option (and the rest of this section is not further shaded for this option).

17219 **Scheduling Policies**

17220 The scheduling semantics described in this volume of POSIX.1-2008 are defined in terms of a  
 17221 conceptual model that contains a set of thread lists. No implementation structures are  
 17222 necessarily implied by the use of this conceptual model. It is assumed that no time elapses  
 17223 during operations described using this model, and therefore no simultaneous operations are  
 17224 possible. This model discusses only processor scheduling for runnable threads, but it should be  
 17225 noted that greatly enhanced predictability of realtime applications results if the sequencing of  
 17226 other resources takes processor scheduling policy into account.

17227 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread  
 17228 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty  
 17229 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose  
 17230 of a scheduling policy is to define the allowable operations on this set of lists (for example,  
 17231 moving threads between and within lists).

17232 The POSIX model treats a "process" as an aggregation of system resources, including one or  
 17233 more threads that may be scheduled by the operating system on the processor(s) it controls.  
 17234 Although a process has its own set of scheduling attributes, these have an indirect effect (if any)  
 17235 on the scheduling behavior of individual threads as described below.

17236 Each thread shall be controlled by an associated scheduling policy and priority. These  
 17237 parameters may be specified by explicit application execution of the *pthread\_setschedparam()*  
 17238 function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may  
 17239 be changed by application execution of the *pthread\_setschedprio()* function.

17240 Each process shall be controlled by an associated scheduling policy and priority. These  
 17241 parameters may be specified by explicit application execution of the *sched\_setscheduler()* or  
 17242 *sched\_setparam()* functions.

17243 The effect of the process scheduling attributes on individual threads in the process is dependent  
 17244 on the scheduling contention scope of the threads (see [Section 2.9.4](#), on page 509):

- 17245 • For threads with system scheduling contention scope, the process scheduling attributes  
 17246 shall have no effect on the scheduling attributes or behavior either of the thread or an  
 17247 underlying kernel scheduling entity dedicated to that thread.
- 17248 • For threads with process scheduling contention scope, the process scheduling attributes  
 17249 shall have no effect on the scheduling attributes of the thread. However, any underlying  
 17250 kernel scheduling entity used by these threads shall at all times behave as specified by the  
 17251 scheduling attributes of the containing process, and this behavior may affect the  
 17252 scheduling behavior of the process contention scope threads. For example, a process  
 17253 contention scope thread with scheduling policy `SCHED_FIFO` and the system maximum  
 17254 priority *H* (the value returned by *sched\_get\_priority\_max(SCHED\_FIFO)*) in a process with  
 17255 scheduling policy `SCHED_RR` and system minimum priority *L* (the value returned by  
 17256 *sched\_get\_priority\_min(SCHED\_RR)*) shall be subject to timeslicing and to preemption by  
 17257 any thread with an effective priority higher than *L*.

17258 Associated with each policy is a priority range. Each policy definition shall specify the minimum  
 17259 priority range for that policy. The priority ranges for each policy may but need not overlap the  
 17260 priority ranges of other policies.

17261 A conforming implementation shall select the thread that is defined as being at the head of the

- 17262 highest priority non-empty thread list to become a running thread, regardless of its associated  
17263 policy. This thread is then removed from its thread list.
- 17264 Four scheduling policies are specifically required. Other implementation-defined scheduling  
17265 policies may be defined. The following symbols are defined in the Base Definitions volume of  
17266 POSIX.1-2008, <sched.h>:
- 17267 SCHED\_FIFO First in, first out (FIFO) scheduling policy.
- 17268 SCHED\_RR Round robin scheduling policy.
- 17269 SS SCHED\_SPORADIC Sporadic server scheduling policy.
- 17270 SCHED\_OTHER Another scheduling policy.
- 17271 The values of these symbols shall be distinct.
- 17272 **SCHED\_FIFO**
- 17273 Conforming implementations shall include a scheduling policy called the FIFO scheduling  
17274 policy.
- 17275 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its  
17276 threads have been on the list without being executed; generally, the head of the list is the thread  
17277 that has been on the list the longest time, and the tail is the thread that has been on the list the  
17278 shortest time.
- 17279 Under the SCHED\_FIFO policy, the modification of the definitional thread lists is as follows:
- 17280 1. When a running thread becomes a preempted thread, it becomes the head of the thread  
17281 list for its priority.
  - 17282 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list  
17283 for its priority.
  - 17284 3. When a running thread calls the *sched\_setscheduler()* function, the process specified in the  
17285 function call is modified to the specified policy and the priority specified by the *param*  
17286 argument.
  - 17287 4. When a running thread calls the *sched\_setparam()* function, the priority of the process  
17288 specified in the function call is modified to the priority specified by the *param* argument.
  - 17289 5. When a running thread calls the *pthread\_setschedparam()* function, the thread specified in  
17290 the function call is modified to the specified policy and the priority specified by the *param*  
17291 argument.
  - 17292 6. When a running thread calls the *pthread\_setschedprio()* function, the thread specified in the  
17293 function call is modified to the priority specified by the *prio* argument.
  - 17294 7. If a thread whose policy or priority has been modified other than by *pthread\_setschedprio()*  
17295 is a running thread or is runnable, it then becomes the tail of the thread list for its new  
17296 priority.
  - 17297 8. If a thread whose priority has been modified by *pthread\_setschedprio()* is a running thread  
17298 or is runnable, the effect on its position in the thread list depends on the direction of the  
17299 modification, as follows:
    - 17300 a. If the priority is raised, the thread becomes the tail of the thread list.
    - 17301 b. If the priority is unchanged, the thread does not change position in the thread list.

- 17302 c. If the priority is lowered, the thread becomes the head of the thread list.
- 17303 9. When a running thread issues the *sched\_yield()* function, the thread becomes the tail of
- 17304 the thread list for its priority.
- 17305 10. At no other time is the position of a thread with this scheduling policy within the thread
- 17306 lists affected.

17307 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*

17308 and *sched\_get\_priority\_min()* functions when SCHED\_FIFO is provided as the parameter.

17309 Conforming implementations shall provide a priority range of at least 32 priorities for this

17310 policy.

### 17311 SCHED\_RR

17312 Conforming implementations shall include a scheduling policy called the “round robin”

17313 scheduling policy. This policy shall be identical to the SCHED\_FIFO policy with the additional

17314 condition that when the implementation detects that a running thread has been executing as a

17315 running thread for a time period of the length returned by the *sched\_rr\_get\_interval()* function or

17316 longer, the thread shall become the tail of its thread list and the head of that thread list shall be

17317 removed and made a running thread.

17318 The effect of this policy is to ensure that if there are multiple SCHED\_RR threads at the same

17319 priority, one of them does not monopolize the processor. An application should not rely only on

17320 the use of SCHED\_RR to ensure application progress among multiple threads if the application

17321 includes threads using the SCHED\_FIFO policy at the same or higher priority levels or

17322 SCHED\_RR threads at a higher priority level.

17323 A thread under this policy that is preempted and subsequently resumes execution as a running

17324 thread completes the unexpired portion of its round robin interval time period.

17325 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*

17326 and *sched\_get\_priority\_min()* functions when SCHED\_RR is provided as the parameter.

17327 Conforming implementations shall provide a priority range of at least 32 priorities for this

17328 policy.

### 17329 SCHED\_SPORADIC

17330 SS|TSP The functionality described in this section shall be provided on implementations that support

17331 the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not

17332 further shaded for these options).

17333 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the

17334 implementation shall include a scheduling policy identified by the value SCHED\_SPORADIC.

17335 The sporadic server policy is based primarily on two time parameters: the replenishment period

17336 and the available execution capacity. The replenishment period is given by the

17337 *sched\_ss\_repl\_period* member of the **sched\_param** structure. The available execution capacity is

17338 initialized to the value given by the *sched\_ss\_init\_budget* member of the same parameter. The

17339 sporadic server policy is identical to the SCHED\_FIFO policy with some additional conditions

17340 that cause the thread’s assigned priority to be switched between the values specified by the

17341 *sched\_priority* and *sched\_ss\_low\_priority* members of the **sched\_param** structure.

17342 The priority assigned to a thread using the sporadic server scheduling policy is determined in

17343 the following manner: if the available execution capacity is greater than zero and the number of

17344 pending replenishment operations is strictly less than *sched\_ss\_max\_repl*, the thread is assigned

17345 the priority specified by *sched\_priority*; otherwise, the assigned priority shall be

17346 *sched\_ss\_low\_priority*. If the value of *sched\_priority* is less than or equal to the value of

17347 *sched\_ss\_low\_priority*, the results are undefined. When active, the thread shall belong to the  
 17348 thread list corresponding to its assigned priority level, according to the mentioned priority  
 17349 assignment. The modification of the available execution capacity and, consequently of the  
 17350 assigned priority, is done as follows:

- 17351 1. When the thread at the head of the *sched\_priority* list becomes a running thread, its  
 17352 execution time shall be limited to at most its available execution capacity, plus the  
 17353 resolution of the execution time clock used for this scheduling policy. This resolution shall  
 17354 be implementation-defined.
- 17355 2. Each time the thread is inserted at the tail of the list associated with *sched\_priority*—  
 17356 because as a blocked thread it became runnable with priority *sched\_priority* or because a  
 17357 replenishment operation was performed—the time at which this operation is done is  
 17358 posted as the *activation\_time*.
- 17359 3. When the running thread with assigned priority equal to *sched\_priority* becomes a  
 17360 preempted thread, it becomes the head of the thread list for its priority, and the execution  
 17361 time consumed is subtracted from the available execution capacity. If the available  
 17362 execution capacity would become negative by this operation, it shall be set to zero.
- 17363 4. When the running thread with assigned priority equal to *sched\_priority* becomes a blocked  
 17364 thread, the execution time consumed is subtracted from the available execution capacity,  
 17365 and a replenishment operation is scheduled, as described in 6 and 7. If the available  
 17366 execution capacity would become negative by this operation, it shall be set to zero.
- 17367 5. When the running thread with assigned priority equal to *sched\_priority* reaches the limit  
 17368 imposed on its execution time, it becomes the tail of the thread list for  
 17369 *sched\_ss\_low\_priority*, the execution time consumed is subtracted from the available  
 17370 execution capacity (which becomes zero), and a replenishment operation is scheduled, as  
 17371 described in 6 and 7.
- 17372 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be  
 17373 replenished, *replenish\_amount*, is set equal to the execution time consumed by the thread  
 17374 since the *activation\_time*. The replenishment is scheduled to occur at *activation\_time* plus  
 17375 *sched\_ss\_repl\_period*. If the scheduled time obtained is before the current time, the  
 17376 replenishment operation is carried out immediately. Several replenishment operations  
 17377 may be pending at the same time, each of which will be serviced at its respective  
 17378 scheduled time. With the above rules, the number of replenishment operations  
 17379 simultaneously pending for a given thread that is scheduled under the sporadic server  
 17380 policy shall not be greater than *sched\_ss\_max\_repl*.
- 17381 7. A replenishment operation consists of adding the corresponding *replenish\_amount* to the  
 17382 available execution capacity at the scheduled time. If, as a consequence of this operation,  
 17383 the execution capacity would become larger than *sched\_ss\_initial\_budget*, it shall be  
 17384 rounded down to a value equal to *sched\_ss\_initial\_budget*. Additionally, if the thread was  
 17385 runnable or running, and had assigned priority equal to *sched\_ss\_low\_priority*, then it  
 17386 becomes the tail of the thread list for *sched\_priority*.

17387 Execution time is defined in XBD Section 3.118 (on page 52).

17388 For this policy, changing the value of a CPU-time clock via *clock\_settime()* shall have no effect on  
 17389 its behavior.

17390 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_min()*  
 17391 and *sched\_get\_priority\_max()* functions when SCHED\_SPORADIC is provided as the parameter.  
 17392 Conforming implementations shall provide a priority range of at least 32 distinct priorities for  
 17393 this policy.

17394 If the scheduling policy of the target process is either SCHED\_FIFO or SCHED\_RR, the  
 17395 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, and *sched\_ss\_init* budget members of the *param*  
 17396 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process  
 17397 is not SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC, the effects of these members are  
 17398 implementation-defined; this case includes the SCHED\_OTHER policy.

#### 17399 SCHED\_OTHER

17400 Conforming implementations shall include one scheduling policy identified as SCHED\_OTHER  
 17401 (which may execute identically with either the FIFO or round robin scheduling policy). The  
 17402 effect of scheduling threads with the SCHED\_OTHER policy in a system in which other threads  
 17403 SS are executing under SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC is implementation-  
 17404 defined.

17405 This policy is defined to allow strictly conforming applications to be able to indicate in a  
 17406 portable manner that they no longer need a realtime scheduling policy.

17407 For threads executing under this policy, the implementation shall use only priorities within the  
 17408 range returned by the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions when  
 17409 SCHED\_OTHER is provided as the parameter.

### 17410 2.8.5 Clocks and Timers

17411 The `<time.h>` header defines the types and manifest constants used by the timing facility.

#### 17412 Time Value Specification Structures

17413 Many of the timing facility functions accept or return time value specifications. A time value  
 17414 structure `timespec` specifies a single time value and includes at least the following members:

| Member Type         | Member Name    | Description  |
|---------------------|----------------|--------------|
| <code>time_t</code> | <i>tv_sec</i>  | Seconds.     |
| <code>long</code>   | <i>tv_nsec</i> | Nanoseconds. |

17418 The *tv\_nsec* member is only valid if greater than or equal to zero, and less than the number of  
 17419 nanoseconds in a second (1 000 million). The time interval described by this structure is ( $tv\_sec * 10^9 + tv\_nsec$ ) nanoseconds.

17421 A time value structure `itimerspec` specifies an initial timer value and a repetition interval for use  
 17422 by the per-process timer functions. This structure includes at least the following members:

| Member Type                  | Member Name        | Description       |
|------------------------------|--------------------|-------------------|
| <code>struct timespec</code> | <i>it_interval</i> | Timer period.     |
| <code>struct timespec</code> | <i>it_value</i>    | Timer expiration. |

17426 If the value described by *it\_value* is non-zero, it indicates the time to or time of the next timer  
 17427 expiration (for relative and absolute timer values, respectively). If the value described by *it\_value*  
 17428 is zero, the timer shall be disarmed.

17429 If the value described by *it\_interval* is non-zero, it specifies an interval which shall be used in  
 17430 reloading the timer when it expires; that is, a periodic timer is specified. If the value described  
 17431 by *it\_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is  
 17432 specified.

17433 **Timer Event Notification Control Block**

17434 Per-process timers may be created that notify the process of timer expirations by queuing a  
 17435 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of  
 17436 POSIX.1-2008, **<signal.h>**, is used in creating such a timer. The **sigevent** structure contains the  
 17437 signal number and an application-specific data value which shall be used when notifying the  
 17438 calling process of timer expiration events.

17439 **Manifest Constants**

17440 The following constants are defined in the Base Definitions volume of POSIX.1-2008, **<time.h>**:

17441 **CLOCK\_REALTIME** The identifier for the system-wide realtime clock.

17442 **TIMER\_ABSTIME** Flag indicating time is absolute with respect to the clock associated  
 17443 with a timer.

17444 **MON** **CLOCK\_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined  
 17445 as a clock whose value cannot be set via *clock\_settime()* and which  
 17446 cannot have backward clock jumps. The maximum possible clock  
 17447 jump is implementation-defined.

17448 **MON** The maximum allowable resolution for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC** clocks  
 17449 and all time services based on these clocks is represented by **{\_POSIX\_CLOCKRES\_MIN}** and  
 17450 shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of  
 17451 resolution for these clocks to provide finer granularity time bases. The actual resolution  
 17452 supported by an implementation for a specific clock is obtained using the *clock\_getres()* function.  
 17453 If the actual resolution supported for a time service based on one of these clocks differs from the  
 17454 resolution supported for that clock, the implementation shall document this difference.

17455 **MON** The minimum allowable maximum value for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC**  
 17456 clocks and all absolute time services based on them is the same as that defined by the ISO C  
 17457 standard for the **time\_t** type. If the maximum value supported by a time service based on one of  
 17458 these clocks differs from the maximum value supported by that clock, the implementation shall  
 17459 document this difference.

17460 **Execution Time Monitoring**

17461 **CPT** If **\_POSIX\_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the  
 17462 clocks described in **Manifest Constants**.

17463 **TCT** If **\_POSIX\_THREAD\_CPUTIME** is defined, thread CPU-time clocks shall be supported.

17464 **CPT|TCT** CPU-time clocks measure execution or CPU time, which is defined in XBD **Section 3.118** (on  
 17465 page 52). The mechanism used to measure execution time is described in XBD **Section 4.10** (on  
 17466 page 110).

17467 **CPT** If **\_POSIX\_CPUTIME** is defined, the following constant of the type **clockid\_t** is defined in  
 17468 **<time.h>**:

17469 **CLOCK\_PROCESS\_CPUTIME\_ID**

17470 When this value of the type **clockid\_t** is used in a *clock()* or *timer\*()* function call, it is  
 17471 interpreted as the identifier of the CPU-time clock associated with the process making the  
 17472 function call.

17473 **TCT** If **\_POSIX\_THREAD\_CPUTIME** is defined, the following constant of the type **clockid\_t** is  
 17474 defined in **<time.h>**:

17475 CLOCK\_THREAD\_CPUTIME\_ID  
 17476 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is  
 17477 interpreted as the identifier of the CPU-time clock associated with the thread making the  
 17478 function call.

## 17479 2.9 Threads

17480 This section defines functionality to support multiple flows of control, called “threads” within a  
 17481 process. For the definition of threads, see XBD Section 3.396 (on page 97).

17482 The specific functional areas covered by threads and their scope include:

- 17483 • Thread management: the creation, control, and termination of multiple flows of control in  
 17484 the same process under the assumption of a common shared address space
- 17485 • Synchronization primitives optimized for tightly coupled operation of multiple control  
 17486 flows in a common, shared address space

### 17487 2.9.1 Thread-Safety

17488 All functions defined by this volume of POSIX.1-2008 shall be thread-safe, except that the  
 17489 following functions<sup>7</sup> need not be thread-safe.

|       |                             |                                 |                              |                                 |
|-------|-----------------------------|---------------------------------|------------------------------|---------------------------------|
| 17490 | <code>asctime()</code>      | <code>ftw()</code>              | <code>getservbyport()</code> | <code>putc_unlocked()</code>    |
| 17491 | <code>basename()</code>     | <code>getc_unlocked()</code>    | <code>getservent()</code>    | <code>putchar_unlocked()</code> |
| 17492 | <code>catgets()</code>      | <code>getchar_unlocked()</code> | <code>getutxent()</code>     | <code>putenv()</code>           |
| 17493 | <code>crypt()</code>        | <code>getdate()</code>          | <code>getutxid()</code>      | <code>pututxline()</code>       |
| 17494 | <code>ctime()</code>        | <code>getenv()</code>           | <code>getutxline()</code>    | <code>rand()</code>             |
| 17495 | <code>dbm_clearerr()</code> | <code>getgrent()</code>         | <code>gmtime()</code>        | <code>readdir()</code>          |
| 17496 | <code>dbm_close()</code>    | <code>getgrgid()</code>         | <code>hcreate()</code>       | <code>setenv()</code>           |
| 17497 | <code>dbm_delete()</code>   | <code>getgrnam()</code>         | <code>hdestroy()</code>      | <code>setgrent()</code>         |
| 17498 | <code>dbm_error()</code>    | <code>gethostent()</code>       | <code>hsearch()</code>       | <code>setkey()</code>           |
| 17499 | <code>dbm_fetch()</code>    | <code>getlogin()</code>         | <code>inet_ntoa()</code>     | <code>setpwent()</code>         |
| 17500 | <code>dbm_firstkey()</code> | <code>getnetbyaddr()</code>     | <code>l64a()</code>          | <code>setutxent()</code>        |
| 17501 | <code>dbm_nextkey()</code>  | <code>getnetbyname()</code>     | <code>lgamma()</code>        | <code>strerror()</code>         |
| 17502 | <code>dbm_open()</code>     | <code>getnetent()</code>        | <code>lgammaf()</code>       | <code>strsignal()</code>        |
| 17503 | <code>dbm_store()</code>    | <code>getopt()</code>           | <code>lgammal()</code>       | <code>strtok()</code>           |
| 17504 | <code>dirname()</code>      | <code>getprotobyname()</code>   | <code>localeconv()</code>    | <code>system()</code>           |
| 17505 | <code>dlderror()</code>     | <code>getprotobynumber()</code> | <code>localtime()</code>     | <code>ttyname()</code>          |
| 17506 | <code>drand48()</code>      | <code>getprotoent()</code>      | <code>lrand48()</code>       | <code>unsetenv()</code>         |
| 17507 | <code>encrypt()</code>      | <code>getpwent()</code>         | <code>mrnd48()</code>        | <code>wcstombs()</code>         |
| 17508 | <code>endgrent()</code>     | <code>getpwnam()</code>         | <code>nftw()</code>          | <code>wctomb()</code>           |
| 17509 | <code>endpwent()</code>     | <code>getpwuid()</code>         | <code>nl_langinfo()</code>   |                                 |
| 17510 | <code>endutxent()</code>    | <code>getservbyname()</code>    | <code>ptsname()</code>       |                                 |

17511 The `ctermid()` and `tmpnam()` functions need not be thread-safe if passed a NULL argument. The  
 17512 `wcrtomb()` and `wcsrtombs()` functions need not be thread-safe if passed a NULL *ps* argument.

17513 7. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

17514 Implementations shall provide internal synchronization as necessary in order to satisfy this  
17515 requirement.

17516 Since multi-threaded applications are not allowed to use the *environ* variable to access or modify  
17517 any environment variable while any other thread is concurrently modifying any environment  
17518 variable, any function dependent on any environment variable is not thread-safe if another  
17519 thread is modifying the environment; see XSH *exec* (on page 772).

### 17520 2.9.2 Thread IDs

17521 Although implementations may have thread IDs that are unique in a system, applications  
17522 should only assume that thread IDs are usable and unique within a single process. The effect of  
17523 calling any of the functions defined in this volume of POSIX.1-2008 and passing as an argument  
17524 the thread ID of a thread from another process is unspecified. The lifetime of a thread ID ends  
17525 after the thread terminates if it was created with the *detachstate* attribute set to  
17526 PTHREAD\_CREATE\_DETACHED or if *pthread\_detach()* or *pthread\_join()* has been called for that  
17527 thread. A conforming implementation is free to reuse a thread ID after its lifetime has ended. If  
17528 an application attempts to use a thread ID whose lifetime has ended, the behavior is undefined.

17529 If a thread is detached, its thread ID is invalid for use as an argument in a call to *pthread\_detach()*  
17530 or *pthread\_join()*.

### 17531 2.9.3 Thread Mutexes

17532 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same  
17533 processing resources from eventually making forward progress in its execution. Eligibility for  
17534 processing resources is determined by the scheduling policy.

17535 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- 17536 • It returns successfully from *pthread\_mutex\_lock()* with *m* as the *mutex* argument.
- 17537 • It returns successfully from *pthread\_mutex\_trylock()* with *m* as the *mutex* argument.
- 17538 • It returns successfully from *pthread\_mutex\_timedlock()* with *m* as the *mutex* argument.
- 17539 • It returns (successfully or not) from *pthread\_cond\_wait()* with *m* as the *mutex* argument  
17540 (except as explicitly indicated otherwise for certain errors).
- 17541 • It returns (successfully or not) from *pthread\_cond\_timedwait()* with *m* as the *mutex*  
17542 argument (except as explicitly indicated otherwise for certain errors).

17543 The thread shall remain the owner of *m* until one of the following occurs:

- 17544 • It executes *pthread\_mutex\_unlock()* with *m* as the *mutex* argument
- 17545 • It blocks in a call to *pthread\_cond\_wait()* with *m* as the *mutex* argument.
- 17546 • It blocks in a call to *pthread\_cond\_timedwait()* with *m* as the *mutex* argument.

17547 The implementation shall behave as if at all times there is at most one owner of any mutex.

17548 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex  
17549 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have  
17550 “released” the mutex and the mutex is said to have become “unlocked”.

17551 A problem can occur if a process terminates while one of its threads holds a mutex lock.  
17552 Depending on the mutex type, it might be possible for another thread to unlock the mutex and  
17553 recover the state of the mutex. However, it is difficult to perform this recovery reliably.

17554 Robust mutexes provide a means to enable the implementation to notify other threads in the  
 17555 event of a process terminating while one of its threads holds a mutex lock. The next thread that  
 17556 acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from  
 17557 the locking function. The notified thread can then attempt to recover the state protected by the  
 17558 mutex, and if successful mark the state protected by the mutex as consistent by a call to  
 17559 *pthread\_mutex\_consistent*( ). If the notified thread is unable to recover the state, it can declare the  
 17560 state as not recoverable by a call to *pthread\_mutex\_unlock*( ) without a prior call to  
 17561 *pthread\_mutex\_consistent*( ).

17562 Whether or not the state protected by a mutex can be recovered is dependent solely on the  
 17563 application using robust mutexes. The robust mutex support provided in the implementation  
 17564 provides notification only that a mutex owner has terminated while holding a lock, or that the  
 17565 state of the mutex is not recoverable.

#### 17566 2.9.4 Thread Scheduling

17567 TPS The functionality described in this section shall be provided on implementations that support  
 17568 the Thread Execution Scheduling option (and the rest of this section is not further shaded for  
 17569 this option).

##### 17570 Thread Scheduling Attributes

17571 In support of the scheduling function, threads have attributes which are accessed through the  
 17572 **pthread\_attr\_t** thread creation attributes object.

17573 The *contentionscope* attribute defines the scheduling contention scope of the thread to be either  
 17574 PTHREAD\_SCOPE\_PROCESS or PTHREAD\_SCOPE\_SYSTEM.

17575 The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling  
 17576 attributes of the creating thread or to have its scheduling values set according to the other  
 17577 scheduling attributes in the **pthread\_attr\_t** object.

17578 The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute  
 17579 defines the scheduling parameters for the thread. The interaction of threads having different  
 17580 policies within a process is described as part of the definition of those policies.

17581 If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one  
 17582 of the priority-based policies defined under this option, the *schedparam* attribute contains the  
 17583 scheduling priority of the thread. A conforming implementation ensures that the priority value  
 17584 in *schedparam* is in the range associated with the scheduling policy when the thread attributes  
 17585 object is used to create a thread, or when the scheduling attributes of a thread are dynamically  
 17586 modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

17587 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four  
 17588 new members that are used for the sporadic server scheduling policy. These members are  
 17589 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and *sched\_ss\_max\_repl*. The  
 17590 meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on  
 17591 page 501).

17592 When a process is created, its single thread has a scheduling policy and associated attributes  
 17593 equal to the policy and attributes of the process. The default scheduling contention scope value  
 17594 is implementation-defined. The default values of other scheduling attributes are  
 17595 implementation-defined.

17596 **Thread Scheduling Contention Scope**

17597 The scheduling contention scope of a thread defines the set of threads with which the thread  
 17598 competes for use of the processing resources. The scheduling operation selects at most one  
 17599 thread to execute on each processor at any point in time and the thread's scheduling attributes  
 17600 (for example, *priority*), whether under process scheduling contention scope or system scheduling  
 17601 contention scope, are the parameters used to determine the scheduling decision.

17602 The scheduling contention scope, in the context of scheduling a mixed scope environment  
 17603 affects threads as follows:

- 17604 • A thread created with PTHREAD\_SCOPE\_SYSTEM scheduling contention scope contends  
 17605 for resources with all other threads in the same scheduling allocation domain relative to  
 17606 their system scheduling attributes. The system scheduling attributes of a thread created  
 17607 with PTHREAD\_SCOPE\_SYSTEM scheduling contention scope are the scheduling  
 17608 attributes with which the thread was created. The system scheduling attributes of a thread  
 17609 created with PTHREAD\_SCOPE\_PROCESS scheduling contention scope are the  
 17610 implementation-defined mapping into system attribute space of the scheduling attributes  
 17611 with which the thread was created.
- 17612 • Threads created with PTHREAD\_SCOPE\_PROCESS scheduling contention scope contend  
 17613 directly with other threads within their process that were created with  
 17614 PTHREAD\_SCOPE\_PROCESS scheduling contention scope. The contention is resolved  
 17615 based on the threads' scheduling attributes and policies. It is unspecified how such threads  
 17616 are scheduled relative to threads in other processes or threads with  
 17617 PTHREAD\_SCOPE\_SYSTEM scheduling contention scope.
- 17618 • Conforming implementations shall support the PTHREAD\_SCOPE\_PROCESS scheduling  
 17619 contention scope, the PTHREAD\_SCOPE\_SYSTEM scheduling contention scope, or both.

17620 **Scheduling Allocation Domain**

17621 Implementations shall support scheduling allocation domains containing one or more  
 17622 processors. It should be noted that the presence of multiple processors does not automatically  
 17623 indicate a scheduling allocation domain size greater than one. Conforming implementations on  
 17624 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation  
 17625 domains, and could define these scheduling allocation domains on a per-thread, per-process, or  
 17626 per-system basis, depending on the types of applications intended to be supported by the  
 17627 implementation. The scheduling allocation domain is independent of scheduling contention  
 17628 scope, as the scheduling contention scope merely defines the set of threads with which a thread  
 17629 contends for processor resources, while scheduling allocation domain defines the set of  
 17630 processors for which it contends. The semantics of how this contention is resolved among  
 17631 threads for processors is determined by the scheduling policies of the threads.

17632 The choice of scheduling allocation domain size and the level of application control over  
 17633 scheduling allocation domains is implementation-defined. Conforming implementations may  
 17634 change the size of scheduling allocation domains and the binding of threads to scheduling  
 17635 allocation domains at any time.

17636 For application threads with scheduling allocation domains of size equal to one, the scheduling  
 17637 rules defined for SCHED\_FIFO and SCHED\_RR shall be used; see [Scheduling Policies](#) (on page  
 17638 501). All threads with system scheduling contention scope, regardless of the processes in which  
 17639 they reside, compete for the processor according to their priorities. Threads with process  
 17640 scheduling contention scope compete only with other threads with process scheduling  
 17641 contention scope within their process.

17642 For application threads with scheduling allocation domains of size greater than one, the rules

17643 TSP defined for SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC shall be used in an  
 17644 implementation-defined manner. Each thread with system scheduling contention scope  
 17645 competes for the processors in its scheduling allocation domain in an implementation-defined  
 17646 manner according to its priority. Threads with process scheduling contention scope are  
 17647 scheduled relative to other threads within the same scheduling contention scope in the process.

17648 TSP If \_POSIX\_THREAD\_SPORADIC\_SERVER is defined, the rules defined for SCHED\_SPORADIC  
 17649 in *Scheduling Policies* (on page 501) shall be used in an implementation-defined manner for  
 17650 application threads whose scheduling allocation domain size is greater than one.

### 17651 Scheduling Documentation

17652 If \_POSIX\_PRIORITY\_SCHEDULING is defined, then any scheduling policies beyond  
 17653 TSP SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC, as well as the effects of  
 17654 the scheduling policies indicated by these other values, and the attributes required in order to  
 17655 support such a policy, are implementation-defined. Furthermore, the implementation shall  
 17656 document the effect of all processor scheduling allocation domain values supported for these  
 17657 policies.

## 17658 2.9.5 Thread Cancellation

17659 The thread cancellation mechanism allows a thread to terminate the execution of any other  
 17660 thread in the process in a controlled manner. The target thread (that is, the one that is being  
 17661 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform  
 17662 application-specific cleanup processing when the notice of cancellation is acted upon.

17663 Cancellation is controlled by the cancellation control functions. Each thread maintains its own  
 17664 cancelability state. Cancellation may only occur at cancellation points or when the thread is  
 17665 asynchronously cancelable.

17666 The thread cancellation mechanism described in this section depends upon programs having set  
 17667 *deferred* cancelability state, which is specified as the default. Applications shall also carefully  
 17668 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,  
 17669 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary  
 17670 scope pop operation results in undefined behavior.

17671 Use of asynchronous cancelability while holding resources which potentially need to be released  
 17672 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated  
 17673 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

### 17674 2.9.5.1 Cancelability States

17675 The cancelability state of a thread determines the action taken upon receipt of a cancellation  
 17676 request. The thread may control cancellation in a number of ways.

17677 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 17678 1. Cancelability-Enable: When cancelability is PTHREAD\_CANCEL\_DISABLE (as defined  
 17679 in the Base Definitions volume of POSIX.1-2008, **<pthread.h>**), cancellation requests  
 17680 against the target thread are held pending. By default, cancelability is set to  
 17681 PTHREAD\_CANCEL\_ENABLE (as defined in **<pthread.h>**).
- 17682 2. Cancelability Type: When cancelability is enabled and the cancelability type is  
 17683 PTHREAD\_CANCEL\_ASYNCHRONOUS (as defined in **<pthread.h>**), new or pending  
 17684 cancellation requests may be acted upon at any time. When cancelability is enabled and  
 17685 the cancelability type is PTHREAD\_CANCEL\_DEFERRED (as defined in **<pthread.h>**),

17686 cancellation requests are held pending until a cancellation point (see below) is reached. If  
 17687 cancelability is disabled, the setting of the cancelability type has no immediate effect as all  
 17688 cancellation requests are held pending; however, once cancelability is enabled again the  
 17689 new type is in effect. The cancelability type is PTHREAD\_CANCEL\_DEFERRED in all  
 17690 newly created threads including the thread in which *main()* was first invoked.

17691 2.9.5.2 Cancellation Points

17692 Cancellation points shall occur when a thread is executing the following functions:

|       |                          |                                 |                        |
|-------|--------------------------|---------------------------------|------------------------|
| 17693 | <i>accept()</i>          | <i>nanosleep()</i>              | <i>select()</i>        |
| 17694 | <i>aio_suspend()</i>     | <i>open()</i>                   | <i>sem_timedwait()</i> |
| 17695 | <i>clock_nanosleep()</i> | <i>openat()</i>                 | <i>sem_wait()</i>      |
| 17696 | <i>close()</i>           | <i>pause()</i>                  | <i>send()</i>          |
| 17697 | <i>connect()</i>         | <i>poll()</i>                   | <i>sendmsg()</i>       |
| 17698 | <i>creat()</i>           | <i>pread()</i>                  | <i>sendto()</i>        |
| 17699 | <i>fcntl()†</i>          | <i>pselect()</i>                | <i>sigsuspend()</i>    |
| 17700 | <i>fdatasync()</i>       | <i>pthread_cond_timedwait()</i> | <i>sigtimedwait()</i>  |
| 17701 | <i>fsync()</i>           | <i>pthread_cond_wait()</i>      | <i>sigwait()</i>       |
| 17702 | <i>getmsg()</i>          | <i>pthread_join()</i>           | <i>sigwaitinfo()</i>   |
| 17703 | <i>getpmsg()</i>         | <i>pthread_testcancel()</i>     | <i>sleep()</i>         |
| 17704 | <i>lockf()††</i>         | <i>putmsg()</i>                 | <i>system()</i>        |
| 17705 | <i>mq_receive()</i>      | <i>putpmsg()</i>                | <i>tcdrain()</i>       |
| 17706 | <i>mq_send()</i>         | <i>pwrite()</i>                 | <i>wait()</i>          |
| 17707 | <i>mq_timedreceive()</i> | <i>read()</i>                   | <i>waitid()</i>        |
| 17708 | <i>mq_timedsend()</i>    | <i>readv()</i>                  | <i>waitpid()</i>       |
| 17709 | <i>msgrcv()</i>          | <i>recv()</i>                   | <i>write()</i>         |
| 17710 | <i>msgsnd()</i>          | <i>recvfrom()</i>               | <i>writen()</i>        |
| 17711 | <i>msync()</i>           | <i>recvmsg()</i>                |                        |

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

17712 † When the *cmd* argument is F\_SETLKW.

17713 †† When the *function* argument is F\_LOCK.

17714 A cancellation point may also occur when a thread is executing the following functions:

|       |                      |                           |                         |
|-------|----------------------|---------------------------|-------------------------|
| 17715 | <i>access()</i>      | <i>fprintf()</i>          | <i>getprotobyname()</i> |
| 17716 | <i>asctime()</i>     | <i>fputc()</i>            | <i>getprotoent()</i>    |
| 17717 | <i>asctime_r()</i>   | <i>fputs()</i>            | <i>getpwent()</i>       |
| 17718 | <i>catclose()</i>    | <i>fputwc()</i>           | <i>getpwnam()</i>       |
| 17719 | <i>catgets()</i>     | <i>fputws()</i>           | <i>getpwnam_r()</i>     |
| 17720 | <i>catopen()</i>     | <i>fread()</i>            | <i>getpwuid()</i>       |
| 17721 | <i>chmod()</i>       | <i>freopen()</i>          | <i>getpwuid_r()</i>     |
| 17722 | <i>chown()</i>       | <i>fscanf()</i>           | <i>gets()</i>           |
| 17723 | <i>closedir()</i>    | <i>fseek()</i>            | <i>getserobyname()</i>  |
| 17724 | <i>closelog()</i>    | <i>fseeko()</i>           | <i>getserobyport()</i>  |
| 17725 | <i>ctermid()</i>     | <i>fsetpos()</i>          | <i>getservent()</i>     |
| 17726 | <i>ctime()</i>       | <i>fstat()</i>            | <i>getutxent()</i>      |
| 17727 | <i>ctime_r()</i>     | <i>fstatat()</i>          | <i>getutxid()</i>       |
| 17728 | <i>dbm_close()</i>   | <i>ftell()</i>            | <i>getutxline()</i>     |
| 17729 | <i>dbm_delete()</i>  | <i>ftello()</i>           | <i>getwc()</i>          |
| 17730 | <i>dbm_fetch()</i>   | <i>ftw()</i>              | <i>getwchar()</i>       |
| 17731 | <i>dbm_nextkey()</i> | <i>futimens()</i>         | <i>glob()</i>           |
| 17732 | <i>dbm_open()</i>    | <i>fwprintf()</i>         | <i>iconv_close()</i>    |
| 17733 | <i>dbm_store()</i>   | <i>fwrite()</i>           | <i>iconv_open()</i>     |
| 17734 | <i>dlclose()</i>     | <i>fwscanf()</i>          | <i>ioctl()</i>          |
| 17735 | <i>dlopen()</i>      | <i>getaddrinfo()</i>      | <i>link()</i>           |
| 17736 | <i>dprintf()</i>     | <i>getc()</i>             | <i>linkat()</i>         |
| 17737 | <i>endgrent()</i>    | <i>getc_unlocked()</i>    | <i>lio_listio()</i>     |
| 17738 | <i>endhostent()</i>  | <i>getchar()</i>          | <i>localtime()</i>      |
| 17739 | <i>endnetent()</i>   | <i>getchar_unlocked()</i> | <i>localtime_r()</i>    |
| 17740 | <i>endprotoent()</i> | <i>getcwd()</i>           | <i>lockf()</i>          |
| 17741 | <i>endpwent()</i>    | <i>getdate()</i>          | <i>lseek()</i>          |
| 17742 | <i>endservent()</i>  | <i>getdelim()</i>         | <i>lstat()</i>          |
| 17743 | <i>endutxent()</i>   | <i>getgrent()</i>         | <i>mkdir()</i>          |
| 17744 | <i>faccessat()</i>   | <i>getgrgid()</i>         | <i>mkdirat()</i>        |
| 17745 | <i>fchmod()</i>      | <i>getgrgid_r()</i>       | <i>mkdtemp()</i>        |
| 17746 | <i>fchmodat()</i>    | <i>getgrnam()</i>         | <i>mkfifo()</i>         |
| 17747 | <i>fchown()</i>      | <i>getgrnam_r()</i>       | <i>mkfifoat()</i>       |
| 17748 | <i>fchownat()</i>    | <i>gethostent()</i>       | <i>mknod()</i>          |
| 17749 | <i>fclose()</i>      | <i>gethostid()</i>        | <i>mknodat()</i>        |
| 17750 | <i>fcntl()†</i>      | <i>gethostname()</i>      | <i>mkstemp()</i>        |
| 17751 | <i>fflush()</i>      | <i>getline()</i>          | <i>mktime()</i>         |
| 17752 | <i>fgetc()</i>       | <i>getlogin()</i>         | <i>nftw()</i>           |
| 17753 | <i>fgetpos()</i>     | <i>getlogin_r()</i>       | <i>opendir()</i>        |
| 17754 | <i>fgets()</i>       | <i>getnameinfo()</i>      | <i>openlog()</i>        |
| 17755 | <i>fgetwc()</i>      | <i>getnetbyaddr()</i>     | <i>pathconf()</i>       |
| 17756 | <i>fgetws()</i>      | <i>getnetbyname()</i>     | <i>pclose()</i>         |
| 17757 | <i>fntmsg()</i>      | <i>getnetent()</i>        | <i>perror()</i>         |
| 17758 | <i>fopen()</i>       | <i>getopt()††</i>         | <i>popen()</i>          |
| 17759 | <i>fpathconf()</i>   | <i>getprotobyname()</i>   | <i>posix_fadvise()</i>  |

|       |                                               |                           |                     |
|-------|-----------------------------------------------|---------------------------|---------------------|
| 17760 | <i>posix_fallocate()</i>                      | <i>putc()</i>             | <i>strerror()</i>   |
| 17761 | <i>posix_madvise()</i>                        | <i>putc_unlocked()</i>    | <i>strerror_r()</i> |
| 17762 | <i>posix_openpt()</i>                         | <i>putchar()</i>          | <i>strftime()</i>   |
| 17763 | <i>posix_spawn()</i>                          | <i>putchar_unlocked()</i> | <i>symlink()</i>    |
| 17764 | <i>posix_spawnnp()</i>                        | <i>puts()</i>             | <i>symlinkat()</i>  |
| 17765 | <i>posix_trace_clear()</i>                    | <i>pututxline()</i>       | <i>sync()</i>       |
| 17766 | <i>posix_trace_close()</i>                    | <i>putwc()</i>            | <i>syslog()</i>     |
| 17767 | <i>posix_trace_create()</i>                   | <i>putwchar()</i>         | <i>tmpfile()</i>    |
| 17768 | <i>posix_trace_create_withlog()</i>           | <i>readdir()</i>          | <i>tmpnam()</i>     |
| 17769 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>readdir_r()</i>        | <i>ttyname()</i>    |
| 17770 | <i>posix_trace_eventtypelist_rewind()</i>     | <i>readlink()</i>         | <i>ttyname_r()</i>  |
| 17771 | <i>posix_trace_flush()</i>                    | <i>readlinkat()</i>       | <i>tzset()</i>      |
| 17772 | <i>posix_trace_get_attr()</i>                 | <i>remove()</i>           | <i>ungetc()</i>     |
| 17773 | <i>posix_trace_get_filter()</i>               | <i>rename()</i>           | <i>ungetwc()</i>    |
| 17774 | <i>posix_trace_get_status()</i>               | <i>renameat()</i>         | <i>unlink()</i>     |
| 17775 | <i>posix_trace_getnext_event()</i>            | <i>rewind()</i>           | <i>unlinkat()</i>   |
| 17776 | <i>posix_trace_open()</i>                     | <i>rewinddir()</i>        | <i>utime()</i>      |
| 17777 | <i>posix_trace_rewind()</i>                   | <i>scandir()</i>          | <i>utimensat()</i>  |
| 17778 | <i>posix_trace_set_filter()</i>               | <i>scanf()</i>            | <i>utimes()</i>     |
| 17779 | <i>posix_trace_shutdown()</i>                 | <i>seekdir()</i>          | <i>vdprintf()</i>   |
| 17780 | <i>posix_trace_timedgetnext_event()</i>       | <i>semop()</i>            | <i>vfprintf()</i>   |
| 17781 | <i>posix_typed_mem_open()</i>                 | <i>setgrent()</i>         | <i>vwprintf()</i>   |
| 17782 | <i>printf()</i>                               | <i>sethostent()</i>       | <i>vprintf()</i>    |
| 17783 | <i>psiginfo()</i>                             | <i>setnetent()</i>        | <i>vwprintf()</i>   |
| 17784 | <i>psignal()</i>                              | <i>setprotoent()</i>      | <i>wcsftime()</i>   |
| 17785 | <i>pthread_rwlock_rdlock()</i>                | <i>setpwent()</i>         | <i>wordexp()</i>    |
| 17786 | <i>pthread_rwlock_timedrdlock()</i>           | <i>setservent()</i>       | <i>wprintf()</i>    |
| 17787 | <i>pthread_rwlock_timedwrlock()</i>           | <i>setutxent()</i>        | <i>wscanf()</i>     |
| 17788 | <i>pthread_rwlock_wrlock()</i>                | <i>sigpause()</i>         |                     |
| 17789 |                                               | <i>stat()</i>             |                     |

17790 An implementation shall not introduce cancellation points into any other functions specified in  
17791 this volume of POSIX.1-2008.

17792 The side-effects of acting upon a cancellation request while suspended during a call of a function  
17793 are the same as the side-effects that may be seen in a single-threaded program when a call to a  
17794 function is interrupted by a signal and the given function returns [EINTR]. Any such side-  
17795 effects occur before any cancellation cleanup handlers are called.

17796 Whenever a thread has cancelability enabled and a cancellation request has been made with that  
17797 thread as the target, and the thread then calls any function that is a cancellation point (such as  
17798 *pthread\_testcancel()* or *read()*), the cancellation request shall be acted upon before the function  
17799 returns. If a thread has cancelability enabled and a cancellation request is made with the thread  
17800 as a target while the thread is suspended at a cancellation point, the thread shall be awakened  
17801 and the cancellation request shall be acted upon. It is unspecified whether the cancellation  
17802 request is acted upon or whether the cancellation request remains pending and the thread  
17803 resumes normal execution if:

- 17804 • The thread is suspended at a cancellation point and the event for which it is waiting occurs

17805 † For any value of the *cmd* argument.

17806 †† If *opterr* is non-zero.

17807           • A specified timeout expired  
17808           before the cancellation request is acted upon.

### 17809 2.9.5.3 Thread Cancellation Cleanup Handlers

17810           Each thread maintains a list of cancellation cleanup handlers. The programmer uses the  
17811 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions to place routines on and remove  
17812 routines from this list.

17813           When a cancellation request is acted upon, or when a thread calls *pthread\_exit()*, the thread first  
17814 disables cancellation by setting its cancelability state to `PTHREAD_CANCEL_DISABLE` and its  
17815 cancelability type to `PTHREAD_CANCEL_DEFERRED`. The cancelability state shall remain set  
17816 to `PTHREAD_CANCEL_DISABLE` until the thread has terminated. The behavior is undefined if  
17817 a cancellation cleanup handler or thread-specific data destructor routine changes the  
17818 cancelability state to `PTHREAD_CANCEL_ENABLE`.

17819           The routines in the thread's list of cancellation cleanup handlers are invoked one by one in LIFO  
17820 sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First  
17821 Out). When the cancellation cleanup handler for a scope is invoked, the storage for that scope  
17822 remains valid. If the last cancellation cleanup handler returns, thread-specific data destructors (if  
17823 any) associated with thread-specific data keys for which the thread has non-NULL values will  
17824 be run, in unspecified order, as described for *pthread\_key\_create()*.

17825           After all cancellation cleanup handlers and thread-specific data destructors have returned,  
17826 thread execution is terminated. If the thread has terminated because of a call to *pthread\_exit()*,  
17827 the *value\_ptr* argument is made available to any threads joining with the target. If the thread has  
17828 terminated by acting on a cancellation request, a status of `PTHREAD_CANCELED` is made  
17829 available to any threads joining with the target. The symbolic constant `PTHREAD_CANCELED`  
17830 expands to a constant expression of type `(void *)` whose value matches no pointer to an object in  
17831 memory nor the value `NULL`.

17832           A side-effect of acting upon a cancellation request while in a condition variable wait is that the  
17833 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread  
17834 is no longer considered to be waiting for the condition and the thread shall not have consumed  
17835 any pending condition signals on the condition.

17836           A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

### 17837 2.9.5.4 Async-Cancel Safety

17838           The *pthread\_cancel()*, *pthread\_setcancelstate()*, and *pthread\_setcanceltype()* functions are defined to  
17839 be async-cancel safe.

17840           No other functions in this volume of POSIX.1-2008 are required to be async-cancel-safe.

## 17841 2.9.6 Thread Read-Write Locks

17842           Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
17843 read-only access to data while allowing only one thread to have exclusive write access at any  
17844 given time. They are typically used to protect data that is read more frequently than it is  
17845 changed.

17846           One or more readers acquire read access to the resource by performing a read lock operation on  
17847 the associated read-write lock. A writer acquires exclusive write access by performing a write  
17848 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and

17849 any other writers.

17850 A thread that has blocked on a read-write lock (for example, has not yet returned from a  
17851 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_wrlock()* call) shall not prevent any unblocked thread  
17852 that is eligible to use the same processing resources from eventually making forward progress in  
17853 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

17854 Read-write locks can be used to synchronize threads in the current process and other processes if  
17855 they are allocated in memory that is writable and shared among the cooperating processes and  
17856 have been initialized for this behavior.

### 17857 2.9.7 Thread Interactions with Regular File Operations

17858 All of the following functions shall be atomic with respect to each other in the effects specified in  
17859 POSIX.1-2008 when they operate on regular files or symbolic links:

|       |                   |                    |                     |                    |                    |
|-------|-------------------|--------------------|---------------------|--------------------|--------------------|
| 17860 | <i>chmod()</i>    | <i>fchownat()</i>  | <i>lseek()</i>      | <i>readv()</i>     | <i>unlink()</i>    |
| 17861 | <i>chown()</i>    | <i>fcntl()</i>     | <i>lstat()</i>      | <i>pwrite()</i>    | <i>unlinkat()</i>  |
| 17862 | <i>close()</i>    | <i>fstat()</i>     | <i>open()</i>       | <i>rename()</i>    | <i>utime()</i>     |
| 17863 | <i>creat()</i>    | <i>fstatat()</i>   | <i>openat()</i>     | <i>renameat()</i>  | <i>utimensat()</i> |
| 17864 | <i>dup2()</i>     | <i>ftruncate()</i> | <i>pread()</i>      | <i>stat()</i>      | <i>utimes()</i>    |
| 17865 | <i>fchmod()</i>   | <i>lchown()</i>    | <i>read()</i>       | <i>symlink()</i>   | <i>write()</i>     |
| 17866 | <i>fchmodat()</i> | <i>link()</i>      | <i>readlink()</i>   | <i>symlinkat()</i> | <i>writetv()</i>   |
| 17867 | <i>fchown()</i>   | <i>linkat()</i>    | <i>readlinkat()</i> | <i>truncate()</i>  |                    |

17868 If two threads each call one of these functions, each call shall either see all of the specified effects  
17869 of the other call, or none of them.

### 17870 2.9.8 Use of Application-Managed Thread Stacks

17871 An “application-managed thread stack” is a region of memory allocated by the application—for  
17872 example, memory returned by the *malloc()* or *mmap()* functions—and designated as a stack  
17873 through the act of passing the address and size of the stack, respectively, as the *stackaddr* and  
17874 *stacksize* arguments to *pthread\_attr\_setstack()*. Application-managed stacks allow the application  
17875 to precisely control the placement and size of a stack.

17876 The application grants to the implementation permanent ownership of and control over the  
17877 application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has  
17878 been set is used, either by presenting that attribute’s object as the *attr* argument in a call to  
17879 *pthread\_create()* that completes successfully, or by storing a pointer to the attributes object in the  
17880 *sigev\_notify\_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function  
17881 accepting such argument that completes successfully. The application may thereafter utilize the  
17882 memory within the stack only within the normal context of stack usage within or properly  
17883 synchronized with a thread that has been scheduled by the implementation with stack pointer  
17884 value(s) that are within the range of that stack. In particular, the region of memory cannot be  
17885 freed, nor can it be later specified as the stack for another thread.

17886 When specifying an attributes object with an application-managed stack through the  
17887 *sigev\_notify\_attributes* member of a **struct sigevent**, the results are undefined if the requested  
17888 signal is generated multiple times (as for a repeating timer).

17889 Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the  
17890 application retains ownership of and control over the memory allocated to the stack. It may free  
17891 or reuse the memory as long as it either deletes the attributes object, or before using the

17892 attributes object replaces the stack by making an additional call to *pthread\_attr\_setstack()*, that  
 17893 was used originally to designate the stack. There is no mechanism to retract the reference to an  
 17894 application-managed stack by an existing attributes object.

17895 Once an attributes object with an application-managed stack has been used, that attributes object  
 17896 cannot be used again by a subsequent call to *pthread\_create()* or any function accepting a **struct**  
 17897 **sigevent** with *sigev\_notify\_attributes* containing a pointer to the attributes object, without  
 17898 designating an unused application-managed stack by making an additional call to  
 17899 *pthread\_attr\_setstack()*.

## 17900 2.10 Sockets

17901 A socket is an endpoint for communication using the facilities described in this section. A socket  
 17902 is created with a specific socket type, described in [Section 2.10.6](#) (on page 518), and is associated  
 17903 with a specific protocol, detailed in [Section 2.10.3](#). A socket is accessed via a file descriptor  
 17904 obtained when the socket is created.

### 17905 2.10.1 Address Families

17906 All network protocols are associated with a specific address family. An address family provides  
 17907 basic services to the protocol implementation to allow it to function within a specific network  
 17908 environment. These services may include packet fragmentation and reassembly, routing,  
 17909 addressing, and basic transport. An address family is normally comprised of a number of  
 17910 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not  
 17911 required that an address family support all socket types. An address family may contain  
 17912 multiple protocols supporting the same socket abstraction.

17913 [Section 2.10.17](#) (on page 525), [Section 2.10.19](#) (on page 526), and [Section 2.10.20](#) (on page 526),  
 17914 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based  
 17915 on IPv4, and for Internet protocols based on IPv6.

### 17916 2.10.2 Addressing

17917 An address family defines the format of a socket address. All network addresses are described  
 17918 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of  
 17919 POSIX.1-2008, `<sys/socket.h>`. However, each address family imposes finer and more specific  
 17920 structure, generally defining a structure with fields specific to the address family. The field  
 17921 *sa\_family* in the **sockaddr** structure contains the address family identifier, specifying the format  
 17922 of the *sa\_data* area. The size of the *sa\_data* area is unspecified.

### 17923 2.10.3 Protocols

17924 A protocol supports one of the socket abstractions detailed in [Section 2.10.6](#) (on page 518).  
 17925 Selecting a protocol involves specifying the address family, socket type, and protocol number to  
 17926 the *socket()* function. Certain semantics of the basic socket abstractions are protocol-specific. All  
 17927 protocols are expected to support the basic model for their particular socket type, but may, in  
 17928 addition, provide non-standard facilities or extensions to a mechanism.

17929 **2.10.4 Routing**

17930 Sockets provides packet routing facilities. A routing information database is maintained, which  
17931 is used in selecting the appropriate network interface when transmitting packets.

17932 **2.10.5 Interfaces**

17933 Each network interface in a system corresponds to a path through which messages can be sent  
17934 and received. A network interface usually has a hardware device associated with it, though  
17935 certain interfaces such as the loopback interface, do not.

17936 **2.10.6 Socket Types**

17937 A socket is created with a specific type, which defines the communication semantics and which  
17938 allows the selection of an appropriate communication protocol. Four types are defined:  
17939 RS SOCK\_DGRAM, SOCK\_RAW, SOCK\_SEQPACKET, and SOCK\_STREAM. Implementations  
17940 may specify additional socket types.

17941 The SOCK\_STREAM socket type provides reliable, sequenced, full-duplex octet streams  
17942 between the socket and a peer to which the socket is connected. A socket of type  
17943 SOCK\_STREAM must be in a connected state before any data may be sent or received. Record  
17944 boundaries are not maintained; data sent on a stream socket using output operations of one size  
17945 may be received using input operations of smaller or larger sizes without loss of data. Data may  
17946 be buffered; successful return from an output function does not imply that the data has been  
17947 delivered to the peer or even transmitted from the local system. If data cannot be successfully  
17948 transmitted within a given time then the connection is considered broken, and subsequent  
17949 operations shall fail. A SIGPIPE signal is raised if a thread attempts to send data on a broken  
17950 stream (one that is no longer connected), except that the signal is suppressed if the  
17951 MSG\_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Support for an out-of-  
17952 band data transmission facility is protocol-specific.

17953 The SOCK\_SEQPACKET socket type is similar to the SOCK\_STREAM type, and is also  
17954 connection-oriented. The only difference between these types is that record boundaries are  
17955 maintained using the SOCK\_SEQPACKET type. A record can be sent using one or more output  
17956 operations and received using one or more input operations, but a single operation never  
17957 transfers parts of more than one record. Record boundaries are visible to the receiver via the  
17958 MSG\_EOR flag in the received message flags returned by the *recvmsg()* function. It is protocol-  
17959 specific whether a maximum record size is imposed.

17960 The SOCK\_DGRAM socket type supports connectionless data transfer which is not necessarily  
17961 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or  
17962 broadcast) in each output operation, and incoming datagrams may be received from multiple  
17963 sources. The source address of each datagram is available when receiving the datagram. An  
17964 application may also pre-specify a peer address, in which case calls to output functions that do  
17965 not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only  
17966 datagrams from that peer shall be received. A datagram must be sent in a single output  
17967 operation, and must be received in a single input operation. The maximum size of a datagram is  
17968 protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams  
17969 may be buffered within the system; thus, a successful return from an output function does not  
17970 guarantee that a datagram is actually sent or received. However, implementations should  
17971 attempt to detect any errors possible before the return of an output function, reporting any error  
17972 by an unsuccessful return value.

17973 RS The SOCK\_RAW socket type is similar to the SOCK\_DGRAM type. It differs in that it is  
 17974 normally used with communication providers that underlie those used for the other socket  
 17975 types. For this reason, the creation of a socket with type SOCK\_RAW shall require appropriate  
 17976 privileges. The format of datagrams sent and received with this socket type generally include  
 17977 specific protocol headers, and the formats are protocol-specific and implementation-defined.

#### 17978 **2.10.7 Socket I/O Mode**

17979 The I/O mode of a socket is described by the O\_NONBLOCK file status flag which pertains to  
 17980 the open file description for the socket. This flag is initially off when a socket is created, but may  
 17981 be set and cleared by the use of the F\_SETFL command of the *fcntl()* function.

17982 When the O\_NONBLOCK flag is set, certain functions that would normally block until they are  
 17983 complete shall return immediately.

17984 The *bind()* function initiates an address assignment and shall return without blocking when  
 17985 O\_NONBLOCK is set; if the socket address cannot be assigned immediately, *bind()* shall return  
 17986 the [EINPROGRESS] error to indicate that the assignment was initiated successfully, but that it  
 17987 has not yet completed.

17988 The *connect()* function initiates a connection and shall return without blocking when  
 17989 O\_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection  
 17990 was initiated successfully, but that it has not yet completed.

17991 Data transfer operations (the *read()*, *write()*, *send()*, and *recv()* functions) shall complete  
 17992 immediately, transfer only as much as is available, and then return without blocking, or return  
 17993 an error indicating that no transfer could be made without blocking.

#### 17994 **2.10.8 Socket Owner**

17995 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or  
 17996 process group ID using the F\_SETOWN command of the *fcntl()* function.

#### 17997 **2.10.9 Socket Queue Limits**

17998 The transmit and receive queue sizes for a socket are set when the socket is created. The default  
 17999 sizes used are both protocol-specific and implementation-defined. The sizes may be changed  
 18000 using the *setsockopt()* function.

#### 18001 **2.10.10 Pending Error**

18002 Errors may occur asynchronously, and be reported to the socket in response to input from the  
 18003 network protocol. The socket stores the pending error to be reported to a user of the socket at the  
 18004 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*  
 18005 operation on the socket, and the pending error is then cleared.

18006 **2.10.11 Socket Receive Queue**

18007 A socket has a receive queue that buffers data when it is received by the system until it is  
 18008 removed by a receive call. Depending on the type of the socket and the communication provider,  
 18009 the receive queue may also contain ancillary data such as the addressing and other protocol data  
 18010 associated with the normal data in the queue, and may contain out-of-band or expedited data.  
 18011 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,  
 18012 and ancillary data in the queue. The description in this section applies to all sockets, even  
 18013 though some elements cannot be present in some instances.

18014 The contents of a receive buffer are logically structured as a series of data segments with  
 18015 associated ancillary data and other information. A data segment may contain normal data or  
 18016 out-of-band data, but never both. A data segment may complete a record if the protocol  
 18017 supports records (always true for types SOCK\_SEQPACKET and SOCK\_DGRAM). A record  
 18018 may be stored as more than one segment; the complete record might never be present in the  
 18019 receive buffer at one time, as a portion might already have been returned to the application, and  
 18020 another portion might not yet have been received from the communications provider. A data  
 18021 segment may contain ancillary protocol data, which is logically associated with the segment.  
 18022 Ancillary data is received as if it were queued along with the first normal data octet in the  
 18023 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band  
 18024 data. For the purposes of this section, a datagram is considered to be a data segment that  
 18025 terminates a record, and that includes a source address as a special type of ancillary data. Data  
 18026 segments are placed into the queue as data is delivered to the socket by the protocol. Normal  
 18027 data segments are placed at the end of the queue as they are delivered. If a new segment  
 18028 contains the same type of data as the preceding segment and includes no ancillary data, and if  
 18029 the preceding segment does not terminate a record, the segments are logically merged into a  
 18030 single segment.

18031 The receive queue is logically terminated if an end-of-file indication has been received or a  
 18032 connection has been terminated. A segment shall be considered to be terminated if another  
 18033 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other  
 18034 connection termination has been reported. The last segment in the receive queue shall also be  
 18035 considered to be terminated while the socket has a pending error to be reported.

18036 A receive operation shall never return data or ancillary data from more than one segment.

18037 **2.10.12 Socket Out-of-Band Data State**

18038 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed  
 18039 in the socket receive queue, either at the end of the queue or before all normal data in the queue.  
 18040 In this case, out-of-band data is returned to an application program by a normal receive call.  
 18041 Out-of-band data may also be queued separately rather than being placed in the socket receive  
 18042 queue, in which case it shall be returned only in response to a receive call that requests out-of-  
 18043 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive  
 18044 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An  
 18045 out-of-band data mark is logically an empty data segment that cannot be merged with other  
 18046 segments in the queue. An out-of-band data mark is never returned in response to an input  
 18047 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the  
 18048 first element in the queue. If an out-of-band data mark is the first element in the queue when an  
 18049 input function is called without the MSG\_PEEK option, the mark is removed from the queue  
 18050 and the following data (if any) is processed as if the mark had not been present.

18051 **2.10.13 Connection Indication Queue**

18052 Sockets that are used to accept incoming connections maintain a queue of outstanding  
 18053 connection indications. This queue is a list of connections that are awaiting acceptance by the  
 18054 application; see *listen()*.

18055 **2.10.14 Signals**

18056 One category of event at the socket interface is the generation of signals. These signals report  
 18057 protocol events or process errors relating to the state of the socket. The generation or delivery of  
 18058 a signal does not change the state of the socket, although the generation of the signal may have  
 18059 been caused by a state change.

18060 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no  
 18061 longer able to send (one that is no longer connected), except that the signal is suppressed if the  
 18062 MSG\_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Regardless of whether  
 18063 the generation of the signal is suppressed, the send operation shall fail with the [EPIPE] error.

18064 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified  
 18065 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the  
 18066 status of the socket is specified in [Section 2.10.17](#) (on page 525), [Section 2.10.19](#) (on page 526),  
 18067 and [Section 2.10.20](#) (on page 526). Depending on the protocol, the expedited data may or may  
 18068 not have arrived at the time of signal generation.

18069 **2.10.15 Asynchronous Errors**

18070 If any of the following conditions occur asynchronously for a socket, the corresponding value  
 18071 listed below shall become the pending error for the socket:

18072 [ECONNABORTED]

18073 The connection was aborted locally.

18074 [ECONNREFUSED]

18075 For a connection-mode socket attempting a non-blocking connection, the attempt to connect  
 18076 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram  
 18077 was forcefully rejected.

18078 [ECONNRESET]

18079 The peer has aborted the connection.

18080 [EHOSTDOWN]

18081 The destination host has been determined to be down or disconnected.

18082 [EHOSTUNREACH]

18083 The destination host is not reachable.

18084 [EMSGSIZE]

18085 For a connectionless-mode socket, the size of a previously sent datagram prevented  
 18086 delivery.

18087 [ENETDOWN]

18088 The local network connection is not operational.

18089 [ENETRESET]

18090 The connection was aborted by the network.

18091 [ENETUNREACH]  
 18092 The destination network is not reachable.

18093 **2.10.16 Use of Options**

18094 There are a number of socket options which either specialize the behavior of a socket or provide  
 18095 useful information. These options may be set at different protocol levels and are always present  
 18096 at the uppermost “socket” level.

18097 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions  
 18098 allow an application program to customize the behavior and characteristics of a socket to  
 18099 provide the desired effect.

18100 All of the options have default values. The type and meaning of these values is defined by the  
 18101 protocol level to which they apply. Instead of using the default values, an application program  
 18102 may choose to customize one or more of the options. However, in the bulk of cases, the default  
 18103 values are sufficient for the application.

18104 Some of the options are used to enable or disable certain behavior within the protocol modules  
 18105 (for example, turn on debugging) while others may be used to set protocol-specific information  
 18106 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is  
 18107 introduced, its effect on the underlying protocol modules is described.

18108 Table 2-1 shows the value for the socket level.

18109 **Table 2-1** Value of Level for Socket Options

| Name       | Description                                 |
|------------|---------------------------------------------|
| SOL_SOCKET | Options are intended for the sockets level. |

18112 Table 2-2 (on page 523) lists those options present at the socket level; that is, when the *level*  
 18113 parameter of the *getsockopt()* or *setsockopt()* function is SOL\_SOCKET, the types of the option  
 18114 value parameters associated with each option, and a brief synopsis of the meaning of the option  
 18115 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with  
 18116 *setsockopt()* on all types of socket. Options at other protocol levels vary in format and name.

18117

Table 2-2 Socket-Level Options

| Option        | Parameter Type | Parameter Meaning                                                                                                 |
|---------------|----------------|-------------------------------------------------------------------------------------------------------------------|
| SO_ACCEPTCONN | int            | Non-zero indicates that socket listening is enabled ( <i>getsockopt()</i> only).                                  |
| SO_BROADCAST  | int            | Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).                           |
| SO_DEBUG      | int            | Non-zero requests debugging in underlying protocol modules.                                                       |
| SO_DONTROUTE  | int            | Non-zero requests bypass of normal routing; route based on destination address only.                              |
| SO_ERROR      | int            | Requests and clears pending error information on the socket ( <i>getsockopt()</i> only).                          |
| SO_KEEPALIVE  | int            | Non-zero requests periodic transmission of keepalive messages (protocol-specific).                                |
| SO_LINGER     | struct linger  | Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds. |
| SO_OOBINLINE  | int            | Non-zero requests that out-of-band data be placed into normal data input queue as received.                       |
| SO_RCVBUF     | int            | Size of receive buffer (in bytes).                                                                                |
| SO_RCVLOWAT   | int            | Minimum amount of data to return to application for input operations (in bytes).                                  |
| SO_RCVTIMEO   | struct timeval | Timeout value for a socket receive operation.                                                                     |
| SO_REUSEADDR  | int            | Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).                                  |
| SO_SNDBUF     | int            | Size of send buffer (in bytes).                                                                                   |
| SO_SNDLOWAT   | int            | Minimum amount of data to send for output operations (in bytes).                                                  |
| SO_SNDTIMEO   | struct timeval | Timeout value for a socket send operation.                                                                        |
| SO_TYPE       | int            | Identify socket type ( <i>getsockopt()</i> only).                                                                 |

18148 The SO\_ACCEPTCONN option is used only on *getsockopt()*. When this option is specified,  
 18149 *getsockopt()* shall report whether socket listening is enabled for the socket. A value of zero shall  
 18150 indicate that socket listening is disabled; non-zero that it is enabled. SO\_ACCEPTCONN has no  
 18151 default value.

18152 The SO\_BROADCAST option requests permission to send broadcast datagrams on the socket.  
 18153 Support for SO\_BROADCAST is protocol-specific. The default for SO\_BROADCAST is that the  
 18154 ability to send broadcast datagrams on a socket is disabled.

18155 The SO\_DEBUG option enables debugging in the underlying protocol modules. This can be  
 18156 useful for tracing the behavior of the underlying protocol modules during normal system  
 18157 operation. The semantics of the debug reports are implementation-defined. The default value for  
 18158 SO\_DEBUG is for debugging to be turned off.

18159 The SO\_DONTROUTE option requests that outgoing messages bypass the standard routing  
 18160 facilities. The destination must be on a directly-connected network, and messages are directed to  
 18161 the appropriate network interface according to the destination address. It is protocol-specific  
 18162 whether this option has any effect and how the outgoing network interface is chosen. Support  
 18163 for this option with each protocol is implementation-defined.

18164 The SO\_ERROR option is used only on *getsockopt()*. When this option is specified, *getsockopt()*

18165 shall return any pending error on the socket and clear the error status. It shall return a value of 0  
 18166 if there is no pending error. SO\_ERROR may be used to check for asynchronous errors on  
 18167 connected connectionless-mode sockets or for other types of asynchronous errors. SO\_ERROR  
 18168 has no default value.

18169 The SO\_KEEPALIVE option enables the periodic transmission of messages on a connected  
 18170 socket. The behavior of this option is protocol-specific. On a connection-mode socket for which a  
 18171 connection has been established, if SO\_KEEPALIVE is enabled and the connected socket fails to  
 18172 respond to the keep-alive messages, the connection shall be broken. The default value for  
 18173 SO\_KEEPALIVE is zero, specifying that this capability is turned off.

18174 The SO\_LINGER option controls the action of the interface when unsent messages are queued  
 18175 on a socket and a *close()* is performed. The details of this option are protocol-specific. If  
 18176 SO\_LINGER is enabled, the system shall block the calling thread during *close()* until it can  
 18177 transmit the data or until the end of the interval indicated by the *l\_linger* member, whichever  
 18178 comes first. If SO\_LINGER is not specified, and *close()* is issued, the system handles the call in a  
 18179 way that allows the calling thread to continue as quickly as possible. The default value for  
 18180 SO\_LINGER is zero, or off, for the *l\_onoff* element of the option value and zero seconds for the  
 18181 linger time specified by the *l\_linger* element.

18182 The SO\_OOBINLINE option is valid only on protocols that support out-of-band data. The  
 18183 SO\_OOBINLINE option requests that out-of-band data be placed in the normal data input  
 18184 queue as received; it is then accessible using the *read()* or *recv()* functions without the  
 18185 MSG\_OOB flag set. The default for SO\_OOBINLINE is off; that is, for out-of-band data not to be  
 18186 placed in the normal data input queue.

18187 The SO\_RCVBUF option requests that the buffer space allocated for receive operations on this  
 18188 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer  
 18189 size for high volume connections, or may decrease buffer size to limit the possible backlog of  
 18190 incoming data. The default value for the SO\_RCVBUF option value is implementation-defined,  
 18191 and may vary by protocol.

18192 The SO\_RCVLOWAT option sets the minimum number of bytes to process for socket input  
 18193 operations. In general, receive calls block until any (non-zero) amount of data is received, then  
 18194 return the smaller of the amount available or the amount requested. The default value for  
 18195 SO\_RCVLOWAT is 1, and does not affect the general case. If SO\_RCVLOWAT is set to a larger  
 18196 value, blocking receive calls normally wait until they have received the smaller of the low water  
 18197 mark value or the requested amount. Receive calls may still return less than the low water mark  
 18198 if an error occurs, a signal is caught, or the type of data next in the receive queue is different  
 18199 from that returned (for example, out-of-band data). As mentioned previously, the default value  
 18200 for SO\_RCVLOWAT is 1 byte. It is implementation-defined whether the SO\_RCVLOWAT option  
 18201 can be set.

18202 The SO\_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a  
 18203 **timeval** structure with the number of seconds and microseconds specifying the limit on how  
 18204 long to wait for an input operation to complete. If a receive operation has blocked for this much  
 18205 time without receiving additional data, it shall return with a partial count or *errno* shall be set to  
 18206 [EAGAIN] or [EWOULDBLOCK] if no data were received. The default for this option is the  
 18207 value zero, which indicates that a receive operation will not time out. It is implementation-  
 18208 defined whether the SO\_RCVTIMEO option can be set.

18209 The SO\_REUSEADDR option indicates that the rules used in validating addresses supplied in a  
 18210 *bind()* should allow reuse of local addresses. Operation of this option is protocol-specific. The  
 18211 default value for SO\_REUSEADDR is off; that is, reuse of local addresses is not permitted.

18212 The SO\_SNDBUF option requests that the buffer space allocated for send operations on this  
 18213 socket be set to the value, in bytes, of the option value. The default value for the SO\_SNDBUF

18214 option value is implementation-defined, and may vary by protocol.

18215 The SO\_SNDBLOWAT option sets the minimum number of bytes to process for socket output  
 18216 operations. Most output operations process all of the data supplied by the call, delivering data to  
 18217 the protocol for transmission and blocking as necessary for flow control. Non-blocking output  
 18218 operations process as much data as permitted subject to flow control without blocking, but  
 18219 process no data if flow control does not allow the smaller of the send low water mark value or  
 18220 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall  
 18221 return true only if the send low water mark could be processed. The default value for  
 18222 SO\_SNDBLOWAT is implementation-defined and protocol-specific. It is implementation-defined  
 18223 whether the SO\_SNDBLOWAT option can be set.

18224 The SO\_SNDTIMEO option is an option to set a timeout value for the amount of time that an  
 18225 output function shall block because flow control prevents data from being sent. As noted in  
 18226 Table 2-2 (on page 523), the option value is a **timeval** structure with the number of seconds and  
 18227 microseconds specifying the limit on how long to wait for an output operation to complete. If a  
 18228 send operation has blocked for this much time, it shall return with a partial count or *errno* set to  
 18229 [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is the value  
 18230 zero, which indicates that a send operation will not time out. It is implementation-defined  
 18231 whether the SO\_SNDTIMEO option can be set.

18232 The SO\_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*  
 18233 shall return the type of the socket (for example, SOCK\_STREAM). This option is useful to  
 18234 servers that inherit sockets on start-up. SO\_TYPE has no default value.

## 18235 2.10.17 Use of Sockets for Local UNIX Connections

18236 Support for UNIX domain sockets is mandatory.

18237 UNIX domain sockets provide process-to-process communication in a single system.

### 18238 2.10.17.1 Headers

18239 The symbolic constant AF\_UNIX defined in the `<sys/socket.h>` header is used to identify the  
 18240 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in  
 18241 connection with UNIX domain sockets. See XBD Chapter 13 (on page 219).

18242 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
 18243 accommodate a `sockaddr_un` structure (see the `<sys/un.h>` header defined in XBD Chapter 13,  
 18244 on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be cast as  
 18245 pointers to `sockaddr_un` structures and used to access the fields of those structures without  
 18246 alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_un` structure, the  
 18247 `ss_family` field maps onto the `sun_family` field.

## 18248 2.10.18 Use of Sockets over Internet Protocols

18249 When a socket is created in the Internet family with a protocol value of zero, the implementation  
 18250 shall use the protocol listed below for the type of socket created.

18251 SOCK\_STREAM      IPPROTO\_TCP.

18252 SOCK\_DGRAM      IPPROTO\_UDP.

18253 RS `SOCK_RAW` `IPPROTO_RAW`.

18254 `SOCK_SEQPACKET` Unspecified.

18255 RS A raw interface to IP is available by creating an Internet socket of type `SOCK_RAW`. The default  
 18256 protocol for type `SOCK_RAW` shall be identified in the IP header with the value  
 18257 `IPPROTO_RAW`. Applications should not use the default protocol when creating a socket with  
 18258 type `SOCK_RAW`, but should identify a specific protocol by value. The ICMP control protocol is  
 18259 accessible from a raw socket by specifying a value of `IPPROTO_ICMP` for protocol.

## 18260 2.10.19 Use of Sockets over Internet Protocols Based on IPv4

18261 Support for sockets over Internet protocols based on IPv4 is mandatory.

### 18262 2.10.19.1 Headers

18263 The symbolic constant `AF_INET` defined in the `<sys/socket.h>` header is used to identify the  
 18264 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in  
 18265 connection with IPv4 Internet sockets. See XBD Chapter 13 (on page 219).

18266 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
 18267 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in XBD Chapter  
 18268 13, on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be  
 18269 cast as pointers to `sockaddr_in` structures and used to access the fields of those structures  
 18270 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in`  
 18271 structure, the `ss_family` field maps onto the `sin_family` field.

## 18272 2.10.20 Use of Sockets over Internet Protocols Based on IPv6

18273 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. The  
 18274 functionality described in this section shall be provided on implementations that support the  
 18275 IPV6 option (and the rest of this section is not further shaded for this option).

18276 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain  
 18277 circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 527).

### 18278 2.10.20.1 Addressing

18279 IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead  
 18280 of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

18281 There are three kinds of IPv6 address:

#### 18282 Unicast

18283 Identifies a single interface.

18284 A unicast address can be global, link-local (designed for use on a single link), or site-local  
 18285 (designed for systems not connected to the Internet). Link-local and site-local addresses  
 18286 need not be globally unique.

#### 18287 Anycast

18288 Identifies a set of interfaces such that a packet sent to the address can be delivered to any  
 18289 member of the set.

18290 An anycast address is similar to a unicast address; the nodes to which an anycast address is

- 18291 assigned must be explicitly configured to know that it is an anycast address.
- 18292 Multicast
- 18293 Identifies a set of interfaces such that a packet sent to the address should be delivered to
- 18294 every member of the set.
- 18295 An application can send multicast datagrams by simply specifying an IPv6 multicast
- 18296 address in the *address* argument of *sendto()*. To receive multicast datagrams, an application
- 18297 must join the multicast group (using *setsockopt()* with `IPV6_JOIN_GROUP`) and must bind
- 18298 to the socket the UDP port on which datagrams will be received. Some applications should
- 18299 also bind the multicast group address to the socket, to prevent other datagrams destined to
- 18300 that port from being delivered to the socket.
- 18301 A multicast address can be global, node-local, link-local, site-local, or organization-local.
- 18302 The following special IPv6 addresses are defined:
- 18303 Unspecified
- 18304 An address that is not assigned to any interface and is used to indicate the absence of an
- 18305 address.
- 18306 Loopback
- 18307 A unicast address that is not assigned to any interface and can be used by a node to send
- 18308 packets to itself.
- 18309 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:
- 18310 IPv4-compatible addresses
- 18311 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
- 18312 through IPv4.
- 18313 IPv4-mapped addresses
- 18314 These are used to represent IPv4 addresses in IPv6 address format; see [Section 2.10.20.2](#).
- 18315 Note that the unspecified address and the loopback address must not be treated as
- 18316 IPv4-compatible addresses.
- 18317 **2.10.20.2 Compatibility with IPv4**
- 18318 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
- 18319 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
- 18320 *getaddrinfo()* function when the specified host has only IPv4 addresses.
- 18321 Applications can use `AF_INET6` sockets to open TCP connections to IPv4 nodes, or send UDP
- 18322 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped
- 18323 IPv6 address, and passing that address, within a `sockaddr_in6` structure, in the *connect()*,
- 18324 *sendto()*, or *sendmsg()* function. When applications use `AF_INET6` sockets to accept TCP
- 18325 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return
- 18326 the peer’s address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*
- 18327 function using a `sockaddr_in6` structure encoded this way. If a node has an IPv4 address, then
- 18328 the implementation shall allow applications to communicate using that address via an
- 18329 `AF_INET6` socket. In such a case, the address will be represented at the API by the
- 18330 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an `AF_INET6`
- 18331 socket bound to `in6addr_any` to receive inbound connections and packets destined to one of the
- 18332 node’s IPv4 addresses.
- 18333 An application can use `AF_INET6` sockets to bind to a node’s IPv4 address by specifying the
- 18334 address as an IPv4-mapped IPv6 address in a `sockaddr_in6` structure in the *bind()* function. For
- 18335 an `AF_INET6` socket bound to a node’s IPv4 address, the system shall return the address in the

- 18336 `getsockname()` function as an IPv4-mapped IPv6 address in a `sockaddr_in6` structure.
- 18337 **2.10.20.3 Interface Identification**
- 18338 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;  
18339 zero is not used. There may be gaps so that there is no current interface for a particular positive  
18340 index. Each interface also has a unique implementation-defined name.
- 18341 **2.10.20.4 Options**
- 18342 The following options apply at the IPPROTO\_IPV6 level:
- 18343 **IPV6\_JOIN\_GROUP**  
18344 When set via `setsockopt()`, it joins the application to a multicast group on an interface  
18345 (identified by its index) and addressed by a given multicast address, enabling packets sent  
18346 to that address to be read via the socket. If the interface index is specified as zero, the  
18347 system selects the interface (for example, by looking up the address in a routing table and  
18348 using the resulting interface).
- 18349 An attempt to read this option using `getsockopt()` shall result in an [EOPNOTSUPP] error.
- 18350 The parameter type of this option is a pointer to an `ipv6_mreq` structure.
- 18351 **IPV6\_LEAVE\_GROUP**  
18352 When set via `setsockopt()`, it removes the application from the multicast group on an  
18353 interface (identified by its index) and addressed by a given multicast address.
- 18354 An attempt to read this option using `getsockopt()` shall result in an [EOPNOTSUPP] error.
- 18355 The parameter type of this option is a pointer to an `ipv6_mreq` structure.
- 18356 **IPV6\_MULTICAST\_HOPS**  
18357 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the  
18358 socket. Its possible values are the same as those of IPV6\_UNICAST\_HOPS. If the  
18359 IPV6\_MULTICAST\_HOPS option is not set, a value of 1 is assumed. This option can be set  
18360 via `setsockopt()` and read via `getsockopt()`.
- 18361 The parameter type of this option is a pointer to an `int`. (Default value: 1)
- 18362 **IPV6\_MULTICAST\_IF**  
18363 The index of the interface to be used for outgoing multicast packets. It can be set via  
18364 `setsockopt()` and read via `getsockopt()`. If the interface index is specified as zero, the system  
18365 selects the interface (for example, by looking up the address in a routing table and using the  
18366 resulting interface).
- 18367 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)
- 18368 **IPV6\_MULTICAST\_LOOP**  
18369 This option controls whether outgoing multicast packets should be delivered back to the  
18370 local application when the sending interface is itself a member of the destination multicast  
18371 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an  
18372 [EINVAL] error. This option can be set via `setsockopt()` and read via `getsockopt()`.
- 18373 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean  
18374 value. (Default value: 1)
- 18375 **IPV6\_UNICAST\_HOPS**  
18376 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the  
18377 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to

18378 set a value less than  $-1$  or greater than  $255$  shall result in an [EINVAL] error. This option can  
18379 be set via `setsockopt()` and read via `getsockopt()`.

18380 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

18381 IPV6\_V6ONLY

18382 This socket option restricts AF\_INET6 sockets to IPv6 communications only. AF\_INET6  
18383 sockets may be used for both IPv4 and IPv6 communications. Some applications may want  
18384 to restrict their use of an AF\_INET6 socket to IPv6 communications only. For these  
18385 applications, the IPV6\_V6ONLY socket option is defined. When this option is turned on, the  
18386 socket can be used to send and receive IPv6 packets only. This is an IPPROTO\_IPV6-level  
18387 option.

18388 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.  
18389 (Default value: 0)

18390 An [EOPNOTSUPP] error shall result if IPV6\_JOIN\_GROUP or IPV6\_LEAVE\_GROUP is used  
18391 with `getsockopt()`.

#### 18392 2.10.20.5 Headers

18393 The symbolic constant AF\_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6  
18394 Internet address family. See XBD Chapter 13 (on page 219).

18395 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
18396 accommodate a `sockaddr_in6` structure (see the `<netinet/in.h>` header defined in XBD Chapter  
18397 13, on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be  
18398 cast as pointers to `sockaddr_in6` structures and used to access the fields of those structures  
18399 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in6`  
18400 structure, the `ss_family` field maps onto the `sin6_family` field.

18401 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in  
18402 connection with IPv6 Internet sockets; see XBD Chapter 13 (on page 219).

## 18403 2.11 Tracing

18404 OB TRC This section describes extensions to support tracing of user applications. The functionality  
18405 described in this section is dependent on support of the Trace option (and the rest of this section  
18406 is not further shaded for this option).

18407 The tracing facilities defined in POSIX.1-2008 allow a process to select a set of trace event types,  
18408 to activate a trace stream of the selected trace events as they occur in the flow of execution, and  
18409 to retrieve the recorded trace events.

18410 The tracing operation relies on three logically different components: the traced process, the  
18411 controller process, and the analyzer process. During the execution of the traced process, when a  
18412 trace point is reached, a trace event is recorded into the trace streams created for that process in  
18413 which the associated trace event type identifier is not being filtered out. The controller process  
18414 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 18415 • Initialize the attributes of a trace stream
- 18416 • Create the trace stream (for a specified traced process) using those attributes

- 18417 • Start and stop tracing for the trace stream
- 18418 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 18419 • Shut a trace stream down

18420 These operations can be done for an active trace stream. The analyzer process retrieves the  
 18421 traced events either at runtime, when the trace stream has not yet been shut down, but is still  
 18422 recording trace events; or after opening a trace log that had been previously recorded and shut  
 18423 down. These three logically different operations can be performed by the same process, or can  
 18424 be distributed into different processes.

18425 A trace stream identifier can be created by a call to *posix\_trace\_create()*,  
 18426 *posix\_trace\_create\_withlog()*, or *posix\_trace\_open()*. The *posix\_trace\_create()* and  
 18427 *posix\_trace\_create\_withlog()* functions should be used by a controller process. The  
 18428 *posix\_trace\_open()* should be used by an analyzer process.

18429 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-  
 18430 instrumented code, while another is post-mortem fault analysis. These two potential uses differ  
 18431 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and  
 18432 permits focusing on expected information; while the second needs comprehensive trace  
 18433 capabilities in order to be able to record all types of information.

18434 The events to be traced belong to two classes:

- 18435 1. User trace events (generated by the application instrumentation)
- 18436 2. System trace events (generated by the operating system)

18437 The trace interface defines several system trace event types associated with control of and  
 18438 operation of the trace stream. This small set of system trace events includes the minimum  
 18439 required to interpret correctly the trace event information present in the stream. Other desirable  
 18440 system trace events for some particular application profile may be implemented and are  
 18441 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

18442 Each traced process shall have a mapping of the trace event names to trace event type identifiers  
 18443 that have been defined for that process. Each active trace stream shall have a mapping that  
 18444 incorporates all the trace event type identifiers predefined by the trace system plus all the  
 18445 mappings of trace event names to trace event type identifiers of the processes that are being  
 18446 traced into that trace stream. These mappings are defined from the instrumented application by  
 18447 calling the *posix\_trace\_eventid\_open()* function and from the controller process by calling the  
 18448 *posix\_trace\_trid\_eventid\_open()* function. For a pre-recorded trace stream, the list of trace event  
 18449 types is obtained from the pre-recorded trace log.

18450 The last data modification and file status change timestamps of a file associated with an active  
 18451 trace stream shall be marked for update every time any of the tracing operations modifies that  
 18452 file.

18453 The last data access timestamp of a file associated with a trace stream shall be marked for  
 18454 update every time any of the tracing operations causes data to be read from that file.

18455 Results are undefined if the application performs any operation on a file descriptor associated  
 18456 with an active or pre-recorded trace stream until *posix\_trace\_shutdown()* or *posix\_trace\_close()* is  
 18457 called for that trace stream. Results are also undefined if the analyzer process and the traced  
 18458 process do not share the same programming environment (see *c99*, Programming Environments  
 18459 in the Shell and Utilities volume of POSIX.1-2008).

18460 The main purpose of this option is to define a complete set of functions and concepts that allow  
 18461 a conforming application to be traced from creation to termination, whatever its realtime  
 18462 constraints and properties.

## 18463 2.11.1 Tracing Data Definitions

## 18464 2.11.1.1 Structures

18465 The **<trace.h>** header shall define the *posix\_trace\_status\_info* and *posix\_trace\_event\_info* structures  
18466 described below. Implementations may add extensions to these structures.

18467 **posix\_trace\_status\_info** Structure

18468 To facilitate control of a trace stream, information about the current state of an active trace  
18469 stream can be obtained dynamically. This structure is returned by a call to the  
18470 *posix\_trace\_get\_status()* function.

18471 The **posix\_trace\_status\_info** structure defined in **<trace.h>** shall contain at least the following  
18472 members:

| Member Type | Member Name                        | Description                                                   |
|-------------|------------------------------------|---------------------------------------------------------------|
| int         | <i>posix_stream_status</i>         | The operating mode of the trace stream.                       |
| int         | <i>posix_stream_full_status</i>    | The full status of the trace stream.                          |
| int         | <i>posix_stream_overrun_status</i> | Indicates whether trace events were lost in the trace stream. |

18478 If the Trace Log option is supported in addition to the Trace option, the **posix\_trace\_status\_info**  
18479 structure defined in **<trace.h>** shall contain at least the following additional members:

| Member Type | Member Name                      | Description                                                           |
|-------------|----------------------------------|-----------------------------------------------------------------------|
| int         | <i>posix_stream_flush_status</i> | Indicates whether a flush is in progress.                             |
| int         | <i>posix_stream_flush_error</i>  | Indicates whether any error occurred during the last flush operation. |
| int         | <i>posix_log_overrun_status</i>  | Indicates whether trace events were lost in the trace log.            |
| int         | <i>posix_log_full_status</i>     | The full status of the trace log.                                     |

18487 The *posix\_stream\_status* member indicates the operating mode of the trace stream and shall have  
18488 one of the following values defined by manifest constants in the **<trace.h>** header:

18489 POSIX\_TRACE\_RUNNING

18490 Tracing is in progress; that is, the trace stream is accepting trace events.

18491 POSIX\_TRACE\_SUSPENDED

18492 The trace stream is not accepting trace events. The tracing operation has not yet started or  
18493 has stopped, either following a *posix\_trace\_stop()* function call or because the trace resources  
18494 are exhausted.

18495 The *posix\_stream\_full\_status* member indicates the full status of the trace stream, and it shall have  
18496 one of the following values defined by manifest constants in the **<trace.h>** header:

18497 POSIX\_TRACE\_FULL

18498 The space in the trace stream for trace events is exhausted.

18499 POSIX\_TRACE\_NOT\_FULL

18500 There is still space available in the trace stream.

18501 The combination of the *posix\_stream\_status* and *posix\_stream\_full\_status* members also indicates  
18502 the actual status of the stream. The status shall be interpreted as follows:

|       |                                                                                                                          |
|-------|--------------------------------------------------------------------------------------------------------------------------|
| 18503 | POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL                                                                             |
| 18504 | This status combination indicates that tracing is in progress, and there is space available for                          |
| 18505 | recording more trace events.                                                                                             |
| 18506 | POSIX_TRACE_RUNNING and POSIX_TRACE_FULL                                                                                 |
| 18507 | This status combination indicates that tracing is in progress and that the trace stream is full                          |
| 18508 | of trace events. This status combination cannot occur unless the <i>stream-full-policy</i> is set to                     |
| 18509 | POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving                                        |
| 18510 | time window of prior trace events, and some older trace events may have been overwritten                                 |
| 18511 | and thus lost.                                                                                                           |
| 18512 | POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL                                                                           |
| 18513 | This status combination indicates that tracing has not yet been started, has been stopped by                             |
| 18514 | the <i>posix_trace_stop()</i> function, or has been cleared by the <i>posix_trace_clear()</i> function.                  |
| 18515 | POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL                                                                               |
| 18516 | This status combination indicates that tracing has been stopped by the implementation                                    |
| 18517 | because the <i>stream-full-policy</i> attribute was POSIX_TRACE_UNTIL_FULL and trace                                     |
| 18518 | resources were exhausted, or that the trace stream was stopped by the function                                           |
| 18519 | <i>posix_trace_stop()</i> at a time when trace resources were exhausted.                                                 |
| 18520 | The <i>posix_stream_overrun_status</i> member indicates whether trace events were lost in the trace                      |
| 18521 | stream, and shall have one of the following values defined by manifest constants in the                                  |
| 18522 | <trace.h> header:                                                                                                        |
| 18523 | POSIX_TRACE_OVERRUN                                                                                                      |
| 18524 | At least one trace event was lost and thus was not recorded in the trace stream.                                         |
| 18525 | POSIX_TRACE_NO_OVERRUN                                                                                                   |
| 18526 | No trace events were lost.                                                                                               |
| 18527 | When the corresponding trace stream is created, the <i>posix_stream_overrun_status</i> member shall be                   |
| 18528 | set to POSIX_TRACE_NO_OVERRUN.                                                                                           |
| 18529 | Whenever an overrun occurs, the <i>posix_stream_overrun_status</i> member shall be set to                                |
| 18530 | POSIX_TRACE_OVERRUN.                                                                                                     |
| 18531 | An overrun occurs when:                                                                                                  |
| 18532 | • The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.                                              |
| 18533 | • The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event                                   |
| 18534 | is generated.                                                                                                            |
| 18535 | • If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one                                 |
| 18536 | trace event is lost while flushing the trace stream to the trace log.                                                    |
| 18537 | The <i>posix_stream_overrun_status</i> member is reset to zero after its value is read.                                  |
| 18538 | If the Trace Log option is supported in addition to the Trace option, the <i>posix_stream_flush_status</i> ,             |
| 18539 | <i>posix_stream_flush_error</i> , <i>posix_log_overrun_status</i> , and <i>posix_log_full_status</i> members are defined |
| 18540 | as follows; otherwise, they are undefined.                                                                               |
| 18541 | The <i>posix_stream_flush_status</i> member indicates whether a flush operation is being performed                       |
| 18542 | and shall have one of the following values defined by manifest constants in the header                                   |
| 18543 | <trace.h>:                                                                                                               |
| 18544 | POSIX_TRACE_FLUSHING                                                                                                     |
| 18545 | The trace stream is currently being flushed to the trace log.                                                            |

- 18546 POSIX\_TRACE\_NOT\_FLUSHING  
18547 No flush operation is in progress.
- 18548 The *posix\_stream\_flush\_status* member shall be set to POSIX\_TRACE\_FLUSHING if a flush  
18549 operation is in progress either due to a call to the *posix\_trace\_flush()* function (explicit or caused  
18550 by a trace stream shutdown operation) or because the trace stream has become full with the  
18551 *stream-full-policy* attribute set to POSIX\_TRACE\_FLUSH. The *posix\_stream\_flush\_status* member  
18552 shall be set to POSIX\_TRACE\_NOT\_FLUSHING if no flush operation is in progress.
- 18553 The *posix\_stream\_flush\_error* member shall be set to zero if no error occurred during flushing. If  
18554 an error occurred during a previous flushing operation, the *posix\_stream\_flush\_error* member  
18555 shall be set to the value of the first error that occurred. If more than one error occurs while  
18556 flushing, error values after the first shall be discarded. The *posix\_stream\_flush\_error* member is  
18557 reset to zero after its value is read.
- 18558 The *posix\_log\_overrun\_status* member indicates whether trace events were lost in the trace log,  
18559 and shall have one of the following values defined by manifest constants in the `<trace.h>`  
18560 header:
- 18561 POSIX\_TRACE\_OVERRUN  
18562 At least one trace event was lost.
- 18563 POSIX\_TRACE\_NO\_OVERRUN  
18564 No trace events were lost.
- 18565 When the corresponding trace stream is created, the *posix\_log\_overrun\_status* member shall be set  
18566 to POSIX\_TRACE\_NO\_OVERRUN. Whenever an overrun occurs, this status shall be set to  
18567 POSIX\_TRACE\_OVERRUN. The *posix\_log\_overrun\_status* member is reset to zero after its value  
18568 is read.
- 18569 The *posix\_log\_full\_status* member indicates the full status of the trace log, and it shall have one of  
18570 the following values defined by manifest constants in the `<trace.h>` header:
- 18571 POSIX\_TRACE\_FULL  
18572 The space in the trace log is exhausted.
- 18573 POSIX\_TRACE\_NOT\_FULL  
18574 There is still space available in the trace log.
- 18575 The *posix\_log\_full\_status* member is only meaningful if the *log-full-policy* attribute is either  
18576 POSIX\_TRACE\_UNTIL\_FULL or POSIX\_TRACE\_LOOP.
- 18577 For an active trace stream without log, that is created by the *posix\_trace\_create()* function, the  
18578 *posix\_log\_overrun\_status* member shall be set to POSIX\_TRACE\_NO\_OVERRUN and the  
18579 *posix\_log\_full\_status* member shall be set to POSIX\_TRACE\_NOT\_FULL.
- 18580 **posix\_trace\_event\_info Structure**
- 18581 The trace event structure **posix\_trace\_event\_info** contains the information for one recorded  
18582 trace event. This structure is returned by the set of functions *posix\_trace\_getnext\_event()*,  
18583 *posix\_trace\_timedgetnext\_event()*, and *posix\_trace\_trygetnext\_event()*.
- 18584 The **posix\_trace\_event\_info** structure defined in `<trace.h>` shall contain at least the following  
18585 members:

| Member Type             | Member Name                    | Description                                                               |
|-------------------------|--------------------------------|---------------------------------------------------------------------------|
| <b>trace_event_id_t</b> | <i>posix_event_id</i>          | Trace event type identification.                                          |
| <b>pid_t</b>            | <i>posix_pid</i>               | Process ID of the process that generated the trace event.                 |
| <b>void *</b>           | <i>posix_prog_address</i>      | Address at which the trace point was invoked.                             |
| <b>int</b>              | <i>posix_truncation_status</i> | Status about the truncation of the data associated with this trace event. |
| <b>struct timespec</b>  | <i>posix_timestamp</i>         | Time at which the trace event was generated.                              |

18596 In addition, the **posix\_trace\_event\_info** structure defined in `<trace.h>` shall contain the  
 18597 following additional member:

| Member Type      | Member Name            | Description                                             |
|------------------|------------------------|---------------------------------------------------------|
| <b>pthread_t</b> | <i>posix_thread_id</i> | Thread ID of the thread that generated the trace event. |

18601 The *posix\_event\_id* member represents the identification of the trace event type and its value is  
 18602 not directly defined by the user. This identification is returned by a call to one of the following  
 18603 functions: *posix\_trace\_trid\_eventid\_open()*, *posix\_trace\_eventtypelist\_getnext\_id()*, or  
 18604 *posix\_trace\_eventid\_open()*. The name of the trace event type can be obtained by calling  
 18605 *posix\_trace\_eventid\_get\_name()*.

18606 The *posix\_pid* is the process identifier of the traced process which generated the trace event. If  
 18607 the *posix\_event\_id* member is one of the implementation-defined system trace events and that  
 18608 trace event is not associated with any process, the *posix\_pid* member shall be set to zero.

18609 For a user trace event, the *posix\_prog\_address* member is the process mapped address of the point  
 18610 at which the associated call to the *posix\_trace\_event()* function was made. For a system trace  
 18611 event, if the trace event is caused by a system service explicitly called by the application, the  
 18612 *posix\_prog\_address* member shall be the address of the process at the point where the call to that  
 18613 system service was made.

18614 The *posix\_truncation\_status* member defines whether the data associated with a trace event has  
 18615 been truncated at the time the trace event was generated, or at the time the trace event was read  
 18616 from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event*  
 18617 argument from the *posix\_trace\_getnext\_event()* function). The *posix\_truncation\_status* member  
 18618 shall have one of the following values defined by manifest constants in the `<trace.h>` header:

18619 **POSIX\_TRACE\_NOT\_TRUNCATED**  
 18620 All the traced data is available.

18621 **POSIX\_TRACE\_TRUNCATED\_RECORD**  
 18622 Data was truncated at the time the trace event was generated.

18623 **POSIX\_TRACE\_TRUNCATED\_READ**  
 18624 Data was truncated at the time the trace event was read from a trace stream or a trace log  
 18625 because the reader's buffer was too small. This truncation status overrides the  
 18626 **POSIX\_TRACE\_TRUNCATED\_RECORD** status.

18627 The *posix\_timestamp* member shall be the time at which the trace event was generated. The clock  
 18628 used is implementation-defined, but the resolution of this clock can be retrieved by a call to the  
 18629 *posix\_trace\_attr\_getclockres()* function.

18630 The *posix\_thread\_id* member is the identifier of the thread that generated the trace event. If the  
 18631 *posix\_event\_id* member is one of the implementation-defined system trace events and that trace

18632 event is not associated with any thread, the *posix\_thread\_id* member shall be set to zero.

#### 18633 2.11.1.2 Trace Stream Attributes

18634 Trace streams have attributes that compose the **posix\_trace\_attr\_t** trace stream attributes object.  
18635 This object shall contain at least the following attributes:

- 18636 • The *generation-version* attribute identifies the origin and version of the trace system.
- 18637 • The *trace-name* attribute is a character string defined by the trace controller, and that  
18638 identifies the trace stream.
- 18639 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 18640 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate  
18641 timestamps.
- 18642 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly  
18643 reserved for the trace events.
- 18644 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its  
18645 value is POSIX\_TRACE\_LOOP, POSIX\_TRACE\_UNTIL\_FULL, or POSIX\_TRACE\_FLUSH.
- 18646 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

18647 In addition, if the Trace option and the Trace Inherit option are both supported, the  
18648 **posix\_trace\_attr\_t** trace stream creation attributes object shall contain at least the following  
18649 attributes:

- 18650 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing  
18651 in its parent's process trace stream. It is either POSIX\_TRACE\_INHERITED or  
18652 POSIX\_TRACE\_CLOSE\_FOR\_CHILD.

18653 In addition, if the Trace option and the Trace Log option are both supported, the  
18654 **posix\_trace\_attr\_t** trace stream creation attributes object shall contain at least the following  
18655 attribute:

- 18656 • If the file type corresponding to the trace log supports the POSIX\_TRACE\_LOOP or the  
18657 POSIX\_TRACE\_UNTIL\_FULL policies, the *log-max-size* attribute defines the maximum  
18658 size in bytes of the trace log associated with an active trace stream. Other stream data—for  
18659 example, trace attribute values—shall not be included in this size.
- 18660 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace  
18661 stream to be POSIX\_TRACE\_LOOP, POSIX\_TRACE\_UNTIL\_FULL, or  
18662 POSIX\_TRACE\_APPEND.

## 18663 2.11.2 Trace Event Type Definitions

### 18664 2.11.2.1 System Trace Event Type Definitions

18665 The following system trace event types, defined in the **<trace.h>** header, track the invocation of  
18666 the trace operations:

- 18667 • POSIX\_TRACE\_START shall be associated with a trace start operation.

- 18668
- POSIX\_TRACE\_STOP shall be associated with a trace stop operation.
- 18669
- If the Trace Event Filter option is supported, POSIX\_TRACE\_FILTER shall be associated
- 18670
- with a trace event type filter change operation.
- 18671
- The following system trace event types, defined in the `<trace.h>` header, report operational trace
- 18672
- events:
- 18673
- POSIX\_TRACE\_OVERFLOW shall mark the beginning of a trace overflow condition.
- 18674
- POSIX\_TRACE\_RESUME shall mark the end of a trace overflow condition.
- 18675
- If the Trace Log option is supported, POSIX\_TRACE\_FLUSH\_START shall mark the
- 18676
- beginning of a flush operation.
- 18677
- If the Trace Log option is supported, POSIX\_TRACE\_FLUSH\_STOP shall mark the end of
- 18678
- a flush operation.
- 18679
- If an implementation-defined trace error condition is reported, it shall be marked
- 18680
- POSIX\_TRACE\_ERROR.
- 18681
- The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
- 18682
- information recorded for each trace event, and also on system trace events that indicate the
- 18683
- invocation of trace control operations and trace system operational trace events.
- 18684
- The POSIX\_TRACE\_START and POSIX\_TRACE\_STOP trace events specify the time windows
- 18685
- during which the trace stream is running.
- 18686
- The POSIX\_TRACE\_STOP trace event with an associated data that is equal to zero
- 18687
- indicates a call of the function `posix_trace_stop()`.
- 18688
- The POSIX\_TRACE\_STOP trace event with an associated data that is different from zero
- 18689
- indicates an automatic stop of the trace stream (see the definition of the
- 18690
- `posix_trace_attr_getstreamfullpolicy()` function in `posix_trace_attr_getinherited()`).
- 18691
- The POSIX\_TRACE\_FILTER trace event indicates that a trace event type filter value changed
- 18692
- while the trace stream was running.
- 18693
- The POSIX\_TRACE\_ERROR serves to inform the analyzer process that an implementation-
- 18694
- defined internal error of the trace system occurred.
- 18695
- The POSIX\_TRACE\_OVERFLOW trace event shall be reported with a timestamp equal to the
- 18696
- timestamp of the first trace event overwritten. This is an indication that some generated trace
- 18697
- events have been lost.
- 18698
- The POSIX\_TRACE\_RESUME trace event shall be reported with a timestamp equal to the
- 18699
- timestamp of the first valid trace event reported after the overflow condition ends and shall be
- 18700
- reported before this first valid trace event. This is an indication that the trace system is reliably
- 18701
- recording trace events after an overflow condition.
- 18702
- Each of these trace event types shall be defined by a constant trace event name and a
- 18703
- `trace_event_id_t` constant; trace event data is associated with some of these trace events.
- 18704
- If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
- 18705
- not supported, the following predefined system trace events in Table 2-3 (on page 537) shall be
- 18706
- defined:

18707

**Table 2-3** Trace Option: System Trace Events

| Event Name           | Constant             | Associated Data |
|----------------------|----------------------|-----------------|
|                      |                      | Data Type       |
| posix_trace_error    | POSIX_TRACE_ERROR    | error           |
|                      |                      | <b>int</b>      |
| posix_trace_start    | POSIX_TRACE_START    | None.           |
| posix_trace_stop     | POSIX_TRACE_STOP     | auto            |
|                      |                      | <b>int</b>      |
| posix_trace_overflow | POSIX_TRACE_OVERFLOW | None.           |
| posix_trace_resume   | POSIX_TRACE_RESUME   | None.           |

18717 If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log  
 18718 option is not supported, the following predefined system trace events in Table 2-4 shall be  
 18719 defined:

18720

**Table 2-4** Trace and Trace Event Filter Options: System Trace Events

| Event Name           | Constant             | Associated Data          |
|----------------------|----------------------|--------------------------|
|                      |                      | Data Type                |
| posix_trace_error    | POSIX_TRACE_ERROR    | error                    |
|                      |                      | <b>int</b>               |
| posix_trace_start    | POSIX_TRACE_START    | event_filter             |
|                      |                      | <b>trace_event_set_t</b> |
| posix_trace_stop     | POSIX_TRACE_STOP     | auto                     |
|                      |                      | <b>int</b>               |
| posix_trace_filter   | POSIX_TRACE_FILTER   | old_event_filter         |
|                      |                      | new_event_filter         |
|                      |                      | <b>trace_event_set_t</b> |
| posix_trace_overflow | POSIX_TRACE_OVERFLOW | None.                    |
| posix_trace_resume   | POSIX_TRACE_RESUME   | None.                    |

18734 If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter  
 18735 option is not supported, the following predefined system trace events in Table 2-5 (on page 538)  
 18736 shall be defined:

18737

**Table 2-5** Trace and Trace Log Options: System Trace Events

18738

18739

18740

18741

18742

18743

18744

18745

18746

18747

18748

| Event Name              | Constant                | Associated Data |
|-------------------------|-------------------------|-----------------|
|                         |                         | Data Type       |
| posix_trace_error       | POSIX_TRACE_ERROR       | error           |
|                         |                         | <b>int</b>      |
| posix_trace_start       | POSIX_TRACE_START       | None.           |
| posix_trace_stop        | POSIX_TRACE_STOP        | auto            |
|                         |                         | <b>int</b>      |
| posix_trace_overflow    | POSIX_TRACE_OVERFLOW    | None.           |
| posix_trace_resume      | POSIX_TRACE_RESUME      | None.           |
| posix_trace_flush_start | POSIX_TRACE_FLUSH_START | None.           |
| posix_trace_flush_stop  | POSIX_TRACE_FLUSH_STOP  | None.           |

18749

18750

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 shall be defined:

18751

**Table 2-6** Trace, Trace Log, and Trace Event Filter Options: System Trace Events

18752

18753

18754

18755

18756

18757

18758

18759

18760

18761

18762

18763

18764

18765

18766

| Event Name              | Constant                | Associated Data          |
|-------------------------|-------------------------|--------------------------|
|                         |                         | Data Type                |
| posix_trace_error       | POSIX_TRACE_ERROR       | error                    |
|                         |                         | <b>int</b>               |
| posix_trace_start       | POSIX_TRACE_START       | event_filter             |
|                         |                         | <b>trace_event_set_t</b> |
| posix_trace_stop        | POSIX_TRACE_STOP        | auto                     |
|                         |                         | <b>int</b>               |
| posix_trace_filter      | POSIX_TRACE_FILTER      | old_event_filter         |
|                         |                         | new_event_filter         |
|                         |                         | <b>trace_event_set_t</b> |
| posix_trace_overflow    | POSIX_TRACE_OVERFLOW    | None.                    |
| posix_trace_resume      | POSIX_TRACE_RESUME      | None.                    |
| posix_trace_flush_start | POSIX_TRACE_FLUSH_START | None.                    |
| posix_trace_flush_stop  | POSIX_TRACE_FLUSH_STOP  | None.                    |

18767 2.11.2.2 User Trace Event Type Definitions

18768

18769

18770

18771

18772

The user trace event POSIX\_TRACE\_UNNAMED\_USEREVENT is defined in the <trace.h> header. If the limit of per-process user trace event names represented by {TRACE\_USER\_EVENT\_MAX} has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

18773

The following predefined user trace event in Table 2-7 (on page 539) shall be defined:

18774

Table 2-7 Trace Option: User Trace Event

18775

18776

| Event Name                    | Constant                      |
|-------------------------------|-------------------------------|
| posix_trace_unnamed_userevent | POSIX_TRACE_UNNAMED_USEREVENT |

18777 **2.11.3 Trace Functions**

18778

18779

18780

18781

18782

18783

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

18784

18785

- A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:

18786

18787

18788

18789

18790

18791

18792

18793

18794

18795

18796

18797

- Attribute initialization and destruction (*posix\_trace\_attr\_init()*)
- Identification information manipulation (*posix\_trace\_attr\_getgenversion()*)
- Trace system behavior modification (*posix\_trace\_attr\_getinherited()*)
- Trace stream and trace log size set (*posix\_trace\_attr\_getmaxusereventsize()*)
- Trace stream creation, flush, and shutdown (*posix\_trace\_create()*)
- Trace stream and trace log clear (*posix\_trace\_clear()*)
- Trace event type identifier manipulation (*posix\_trace\_trid\_eventid\_open()*)
- Trace event type identifier list exploration (*posix\_trace\_eventtypelist\_getnext\_id()*)
- Trace event type set manipulation (*posix\_trace\_eventset\_empty()*)
- Trace event type filter set (*posix\_trace\_set\_filter()*)
- Trace stream start and stop (*posix\_trace\_start()*)
- Trace stream information and status read (*posix\_trace\_get\_attr()*)

18798

18799

- A traced process requires functions to instrument trace points:

18800

18801

18802

18803

18804

18805

18806

18807

- Trace event type identifiers definition and trace points insertion (*posix\_trace\_event()*)
- A trace analyzer process requires functions to retrieve information from a trace stream and trace log:
  - Identification information read (*posix\_trace\_attr\_getgenversion()*)
  - Trace system behavior information read (*posix\_trace\_attr\_getinherited()*)
  - Trace stream and trace log size get (*posix\_trace\_attr\_getmaxusereventsize()*)
  - Trace event type identifier manipulation (*posix\_trace\_trid\_eventid\_open()*)
  - Trace event type identifier list exploration (*posix\_trace\_eventtypelist\_getnext\_id()*)
  - Trace log open, rewind, and close (*posix\_trace\_open()*)

- 18808 — Trace stream information and status read (*posix\_trace\_get\_attr()*)
- 18809 — Trace event read (*posix\_trace\_getnext\_event()*)

18810 **2.12 Data Types**

18811 **2.12.1 Defined Types**

18812 All of the data types used by various functions are defined by the implementation. The  
 18813 following table describes some of these types. Other types referenced in the description of a  
 18814 function, not mentioned here, can be found in the appropriate header for that function.

| Defined Type                      | Description                                                                                                                                 |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 18815 <b>cc_t</b>                 | Type used for terminal special characters.                                                                                                  |
| 18816 <b>clock_t</b>              | Integer or real-floating type used for processor times, as defined in the ISO C standard.                                                   |
| 18817                             |                                                                                                                                             |
| 18818                             |                                                                                                                                             |
| 18819 <b>clockid_t</b>            | Used for clock ID type in some timer functions.                                                                                             |
| 18820 <b>dev_t</b>                | Arithmetic type used for device numbers.                                                                                                    |
| 18821 <b>DIR</b>                  | Type representing a directory stream.                                                                                                       |
| 18822 <b>div_t</b>                | Structure type returned by the <i>div()</i> function.                                                                                       |
| 18823 <b>FILE</b>                 | Structure containing information about a file.                                                                                              |
| 18824 <b>glob_t</b>               | Structure type used in pathname pattern matching.                                                                                           |
| 18825 <b>fpos_t</b>               | Type containing all information needed to specify uniquely every position within a file.                                                    |
| 18826                             |                                                                                                                                             |
| 18827 <b>gid_t</b>                | Integer type used for group IDs.                                                                                                            |
| 18828 <b>iconv_t</b>              | Type used for conversion descriptors.                                                                                                       |
| 18829 <b>id_t</b>                 | Integer type used as a general identifier; can be used to contain at least the largest of a <b>pid_t</b> , <b>uid_t</b> , or <b>gid_t</b> . |
| 18830                             |                                                                                                                                             |
| 18831 <b>ino_t</b>                | Unsigned integer type used for file serial numbers.                                                                                         |
| 18832 <b>key_t</b>                | Arithmetic type used for XSI interprocess communication.                                                                                    |
| 18833 <b>ldiv_t</b>               | Structure type returned by the <i>ldiv()</i> function.                                                                                      |
| 18834 <b>mode_t</b>               | Integer type used for file attributes.                                                                                                      |
| 18835 <b>mqd_t</b>                | Used for message queue descriptors.                                                                                                         |
| 18836 <b>nfds_t</b>               | Integer type used for the number of file descriptors.                                                                                       |
| 18837 <b>nlink_t</b>              | Integer type used for link counts.                                                                                                          |
| 18838 <b>off_t</b>                | Signed integer type used for file sizes.                                                                                                    |
| 18839 <b>pid_t</b>                | Signed integer type used for process and process group IDs.                                                                                 |
| 18840 <b>pthread_attr_t</b>       | Used to identify a thread attribute object.                                                                                                 |
| 18841 <b>pthread_cond_t</b>       | Used for condition variables.                                                                                                               |
| 18842 <b>pthread_condattr_t</b>   | Used to identify a condition attribute object.                                                                                              |
| 18843 <b>pthread_key_t</b>        | Used for thread-specific data keys.                                                                                                         |
| 18844 <b>pthread_mutex_t</b>      | Used for mutexes.                                                                                                                           |
| 18845 <b>pthread_mutexattr_t</b>  | Used to identify a mutex attribute object.                                                                                                  |
| 18846 <b>pthread_once_t</b>       | Used for dynamic package initialization.                                                                                                    |
| 18847 <b>pthread_rwlock_t</b>     | Used for read-write locks.                                                                                                                  |
| 18848 <b>pthread_rwlockattr_t</b> | Used for read-write lock attributes.                                                                                                        |
| 18849 <b>pthread_t</b>            | Used to identify a thread.                                                                                                                  |

|       | Defined Type        | Description                                                                                                                                               |
|-------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18850 |                     |                                                                                                                                                           |
| 18851 | <b>ptrdiff_t</b>    | Signed integer type of the result of subtracting two pointers.                                                                                            |
| 18852 | <b>regex_t</b>      | Structure type used in regular expression matching.                                                                                                       |
| 18853 | <b>regmatch_t</b>   | Structure type used in regular expression matching.                                                                                                       |
| 18854 | <b>rlim_t</b>       | Unsigned integer type used for limit values, to which objects of type <b>int</b> and <b>off_t</b> can be cast without loss of value.                      |
| 18855 |                     |                                                                                                                                                           |
| 18856 | <b>sem_t</b>        | Type used in performing semaphore operations.                                                                                                             |
| 18857 | <b>sig_atomic_t</b> | Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.                                      |
| 18858 |                     |                                                                                                                                                           |
| 18859 | <b>sigset_t</b>     | Integer or structure type of an object used to represent sets of signals.                                                                                 |
| 18860 |                     |                                                                                                                                                           |
| 18861 | <b>size_t</b>       | Unsigned integer type used for size of objects.                                                                                                           |
| 18862 | <b>speed_t</b>      | Type used for terminal baud rates.                                                                                                                        |
| 18863 | <b>ssize_t</b>      | Signed integer type used for a count of bytes or an error indication.                                                                                     |
| 18864 |                     |                                                                                                                                                           |
| 18865 | <b>suseconds_t</b>  | Signed integer type used for time in microseconds.                                                                                                        |
| 18866 | <b>tcflag_t</b>     | Type used for terminal modes.                                                                                                                             |
| 18867 | <b>time_t</b>       | Integer or real-floating type used for time in seconds, as defined in the ISO C standard.                                                                 |
| 18868 |                     |                                                                                                                                                           |
| 18869 | <b>timer_t</b>      | Used for timer ID returned by the <i>timer_create()</i> function.                                                                                         |
| 18870 | <b>uid_t</b>        | Integer type used for user IDs.                                                                                                                           |
| 18871 | <b>va_list</b>      | Type used for traversing variable argument lists.                                                                                                         |
| 18872 | <b>wchar_t</b>      | Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales. |
| 18873 |                     |                                                                                                                                                           |
| 18874 |                     |                                                                                                                                                           |
| 18875 | <b>wctype_t</b>     | Scalar type which represents a character class descriptor.                                                                                                |
| 18876 | <b>wint_t</b>       | Integer type capable of storing any valid value of <b>wchar_t</b> or WEOF.                                                                                |
| 18877 |                     |                                                                                                                                                           |
| 18878 | <b>wordexp_t</b>    | Structure type used in word expansion.                                                                                                                    |

### 18879 2.12.2 The char Type

18880 The type **char** is defined as a single byte; see XBD [Chapter 3](#) (on page 33) (Byte and Character).

### 18881 2.12.3 Pointer Types

18882 All function pointer types shall have the same representation as the type pointer to **void**.  
 18883 Conversion of a function pointer to **void \*** shall not alter the representation. A **void \*** value  
 18884 resulting from such a conversion can be converted back to the original function pointer type,  
 18885 using an explicit cast, without loss of information.

18886 **Note:** The ISO C standard does not require this, but it is required for POSIX conformance.

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

18887

Chapter 3

18888

 *System Interfaces*

18889

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

18890

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**FD\_CLR()**18891 **NAME**

18892 FD\_CLR — macros for synchronous I/O multiplexing

18893 **SYNOPSIS**

```
18894     #include <sys/select.h>
18895     void FD_CLR(int fd, fd_set *fdset);
18896     int FD_ISSET(int fd, fd_set *fdset);
18897     void FD_SET(int fd, fd_set *fdset);
18898     void FD_ZERO(fd_set *fdset);
```

18899 **DESCRIPTION**18900 Refer to *pselect()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

18901 **NAME**18902 `_Exit, _exit` — terminate a process18903 **SYNOPSIS**18904 `#include <stdlib.h>`18905 `void _Exit(int status);`18906 `#include <unistd.h>`18907 `void _exit(int status);`18908 **DESCRIPTION**18909 CX For `_Exit()`: The functionality described on this reference page is aligned with the ISO C  
18910 standard. Any conflict between the requirements described here and the ISO C standard is  
18911 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.18912 CX The value of *status* may be 0, `EXIT_SUCCESS`, `EXIT_FAILURE`, or any other value, though only  
18913 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.18914 CX The `_Exit()` and `_exit()` functions shall be functionally equivalent.18915 CX The `_Exit()` and `_exit()` functions shall not call functions registered with `atexit()` nor any  
18916 CX registered signal handlers. Open streams shall not be flushed. Whether open streams are  
18917 closed (without flushing) is implementation-defined. Finally, the calling process shall be  
18918 terminated with the consequences described below.18919 **Consequences of Process Termination**18920 CX **Process termination caused by any reason shall have the following consequences:**18921 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.  
18922 However, functionality relating to the XSI option is shaded.18923 • All of the file descriptors, directory streams, conversion descriptors, and message catalog  
18924 descriptors open in the calling process shall be closed.18925 XSI • If the parent process of the calling process is executing a `wait()`, `waitid()`, or `waitpid()`, and  
18926 has neither set its `SA_NOCLDWAIT` flag nor set `SIGCHLD` to `SIG_IGN`, it shall be notified  
18927 of termination of the calling process and the low-order eight bits (that is, bits 0377) of *status*  
18928 shall be made available to it. If the parent is not waiting, the child's status shall be made  
18929 available to it when the parent subsequently executes `wait()`, `waitid()`, or `waitpid()`.18930 The semantics of the `waitid()` function shall be equivalent to `wait()`.18931 XSI • If the parent process of the calling process is not executing a `wait()`, `waitid()`, or `waitpid()`,  
18932 and has neither set its `SA_NOCLDWAIT` flag nor set `SIGCHLD` to `SIG_IGN`, the calling  
18933 process shall be transformed into a *zombie process*. A *zombie process* is an inactive process  
18934 and it shall be deleted at some later time when its parent process executes `wait()`, `waitid()`,  
18935 or `waitpid()`.18936 XSI **The semantics of the `waitid()` function shall be equivalent to `wait()`.**18937 • Termination of a process does not directly terminate its children. The sending of a `SIGHUP`  
18938 signal as described below indirectly terminates children in some circumstances.

18939 • Either:

18940 If the implementation supports the `SIGCHLD` signal, a `SIGCHLD` shall be sent to the  
18941 parent process.

18942 Or:

Exit()

- 18943 XSI If the parent process has set its SA\_NOCLDWAIT flag, or set SIGCHLD to SIG\_IGN, the  
18944 status shall be discarded, and the lifetime of the calling process shall end immediately. If  
18945 SA\_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to  
18946 the parent process.
- 18947 • The parent process ID of all of the existing child processes and zombie processes of the  
18948 calling process shall be set to the process ID of an implementation-defined system process.  
18949 That is, these processes shall be inherited by a special system process.
- 18950 XSI • Each attached shared-memory segment is detached and the value of *shm\_nattch* (see  
18951 *shmget()*) in the data structure associated with its shared memory ID shall be decremented  
18952 by 1.
- 18953 XSI • For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that  
18954 value shall be added to the *semval* of the specified semaphore.
- 18955 • If the process is a controlling process, the SIGHUP signal shall be sent to each process in  
18956 the foreground process group of the controlling terminal belonging to the calling process.
- 18957 • If the process is a controlling process, the controlling terminal associated with the session  
18958 shall be disassociated from the session, allowing it to be acquired by a new controlling  
18959 process.
- 18960 • If the exit of the process causes a process group to become orphaned, and if any member of  
18961 the newly-orphaned process group is stopped, then a SIGHUP signal followed by a  
18962 SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- 18963 • All open named semaphores in the calling process shall be closed as if by appropriate calls  
18964 to *sem\_close()*.
- 18965 ML • Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be  
18966 removed. If locked pages in the address space of the calling process are also mapped into  
18967 the address spaces of other processes and are locked by those processes, the locks  
18968 established by the other processes shall be unaffected by the call by this process to *\_Exit()*  
18969 or *\_exit()*.
- 18970 • Memory mappings that were created in the process shall be unmapped before the process  
18971 is destroyed.
- 18972 TYM • Any blocks of typed memory that were mapped in the calling process shall be unmapped,  
18973 as if *mummap()* was implicitly called to unmap them.
- 18974 MSG • All open message queue descriptors in the calling process shall be closed as if by  
18975 appropriate calls to *mq\_close()*.
- 18976 • Any outstanding cancelable asynchronous I/O operations may be canceled. Those  
18977 asynchronous I/O operations that are not canceled shall complete as if the *\_Exit()* or  
18978 *\_exit()* operation had not yet occurred, but any associated signal notifications shall be  
18979 suppressed. The *\_Exit()* or *\_exit()* operation may block awaiting such I/O completion.  
18980 Whether any I/O is canceled, and which I/O may be canceled upon *\_Exit()* or *\_exit()*, is  
18981 implementation-defined.
- 18982 • Threads terminated by a call to *\_Exit()* or *\_exit()* shall not invoke their cancellation  
18983 cleanup handlers or per-thread data destructors.
- 18984 OB TRC • If the calling process is a trace controller process, any trace streams that were created by  
18985 the calling process shall be shut down as described by the *posix\_trace\_shutdown()* function,  
18986 and mapping of trace event names to trace event type identifiers of any process built for  
18987 these trace streams may be deallocated.

18988 **RETURN VALUE**

18989 These functions do not return.

18990 **ERRORS**

18991 No errors are defined.

18992 **EXAMPLES**

18993 None.

18994 **APPLICATION USAGE**18995 Normally applications should use *exit()* rather than *\_Exit()* or *\_exit()*.18996 **RATIONALE**18997 **Process Termination**

18998 Early proposals drew a distinction between normal and abnormal process termination.  
 18999 Abnormal termination was caused only by certain signals and resulted in implementation-  
 19000 defined “actions”, as discussed below. Subsequent proposals distinguished three types of  
 19001 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and  
 19002 *abnormal termination with actions*. Again the distinction between the two types of abnormal  
 19003 termination was that they were caused by different signals and that implementation-defined  
 19004 actions would result in the latter case. Given that these actions were completely implementation-  
 19005 defined, the early proposals were only saying when the actions could occur and how their  
 19006 occurrence could be detected, but not what they were. This was of little or no use to conforming  
 19007 applications, and thus the distinction is not made in this volume of POSIX.1-2008.

19008 The implementation-defined actions usually include, in most historical implementations, the  
 19009 creation of a file named **core** in the current working directory of the process. This file contains an  
 19010 image of the memory of the process, together with descriptive information about the process,  
 19011 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

19012 There is a potential security problem in creating a **core** file if the process was set-user-ID and the  
 19013 current user is not the owner of the program, if the process was set-group-ID and none of the  
 19014 user’s groups match the group of the program, or if the user does not have permission to write  
 19015 in the current directory. In this situation, an implementation either should not create a **core** file  
 19016 or should make it unreadable by the user.

19017 Despite the silence of this volume of POSIX.1-2008 on this feature, applications are advised not  
 19018 to create files named **core** because of potential conflicts in many implementations. Some  
 19019 implementations use a name other than **core** for the file; for example, by appending the process  
 19020 ID to the filename.

19021 **Terminating a Process**

19022 It is important that the consequences of process termination as described occur regardless of  
 19023 whether the process called *\_exit()* (perhaps indirectly through *exit()*) or instead was terminated  
 19024 due to a signal or for some other reason. Note that in the specific case of *exit()* this means that  
 19025 the *status* argument to *exit()* is treated in the same way as the *status* argument to *\_exit()*.

19026 A language other than C may have other termination primitives than the C-language *exit()*  
 19027 function, and programs written in such a language should use its native termination primitives,  
 19028 but those should have as part of their function the behavior of *\_exit()* as described.  
 19029 Implementations in languages other than C are outside the scope of this version of this volume  
 19030 of POSIX.1-2008, however.

19031 As required by the ISO C standard, using **return** from *main()* has the same behavior (other than

**\_Exit()**

19032 with respect to language scope issues) as calling *exit()* with the returned value. Reaching the end  
19033 of the *main()* function has the same behavior as calling *exit(0)*.

19034 A value of zero (or *EXIT\_SUCCESS*, which is required to be zero) for the argument *status*  
19035 conventionally indicates successful termination. This corresponds to the specification for *exit()*  
19036 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,  
19037 which interpret a zero status from a child process as success. For this reason, applications should  
19038 not call *exit(0)* or *\_exit(0)* when they terminate unsuccessfully; for example, in signal-catching  
19039 functions.

19040 Historically, the implementation-defined process that inherits children whose parents have  
19041 terminated without waiting on them is called *init* and has a process ID of 1.

19042 The sending of a *SIGHUP* to the foreground process group when a controlling process  
19043 terminates corresponds to somewhat different historical implementations. In System V, the  
19044 kernel sends a *SIGHUP* on termination of (essentially) a controlling process. In 4.2 BSD, the  
19045 kernel does not send *SIGHUP* in a case like this, but the termination of a controlling process is  
19046 usually noticed by a system daemon, which arranges to send a *SIGHUP* to the foreground  
19047 process group with the *vhangup()* function. However, in 4.2 BSD, due to the behavior of the  
19048 shells that support job control, the controlling process is usually a shell with no other processes  
19049 in its process group. Thus, a change to make *\_exit()* behave this way in such systems should not  
19050 cause problems with existing applications.

19051 The termination of a process may cause a process group to become orphaned in either of two  
19052 ways. The connection of a process group to its parent(s) outside of the group depends on both  
19053 the parents and their children. Thus, a process group may be orphaned by the termination of the  
19054 last connecting parent process outside of the group or by the termination of the last direct  
19055 descendant of the parent process(es). In either case, if the termination of a process causes a  
19056 process group to become orphaned, processes within the group are disconnected from their job  
19057 control shell, which no longer has any information on the existence of the process group.  
19058 Stopped processes within the group would languish forever. In order to avoid this problem,  
19059 newly orphaned process groups that contain stopped processes are sent a *SIGHUP* signal and a  
19060 *SIGCONT* signal to indicate that they have been disconnected from their session. The *SIGHUP*  
19061 signal causes the process group members to terminate unless they are catching or ignoring  
19062 *SIGHUP*. Under most circumstances, all of the members of the process group are stopped if any  
19063 of them are stopped.

19064 The action of sending a *SIGHUP* and a *SIGCONT* signal to members of a newly orphaned  
19065 process group is similar to the action of 4.2 BSD, which sends *SIGHUP* and *SIGCONT* to each  
19066 stopped child of an exiting process. If such children exit in response to the *SIGHUP*, any  
19067 additional descendants receive similar treatment at that time. In this volume of POSIX.1-2008,  
19068 the signals are sent to the entire process group at the same time. Also, in this volume of  
19069 POSIX.1-2008, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a  
19070 process group that is not orphaned; therefore, the action taken at *\_exit()* must consider processes  
19071 other than child processes.

19072 It is possible for a process group to be orphaned by a call to *setpgid()* or *setsid()*, as well as by  
19073 process termination. This volume of POSIX.1-2008 does not require sending *SIGHUP* and  
19074 *SIGCONT* in those cases, because, unlike process termination, those cases are not caused  
19075 accidentally by applications that are unaware of job control. An implementation can choose to  
19076 send *SIGHUP* and *SIGCONT* in those cases as an extension; such an extension must be  
19077 documented as required in **<signal.h>**.

19078 The ISO/IEC 9899:1999 standard adds the *\_Exit()* function that results in immediate program  
19079 termination without triggering signals or *atexit()*-registered functions. In POSIX.1-2008, this is

19080 equivalent to the `_exit()` function.

19081 **FUTURE DIRECTIONS**

19082 None.

19083 **SEE ALSO**

19084 `atexit()`, `exit()`, `mlock()`, `mlockall()`, `mq_close()`, `munmap()`, `posix_trace_create()`, `sem_close()`,  
19085 `semop()`, `setpgid()`, `setsid()`, `shmget()`, `wait()`, `waitid()`

19086 XBD `<stdlib.h>`, `<unistd.h>`

19087 **CHANGE HISTORY**

19088 First released in Issue 1. Derived from Issue 1 of the SVID.

19089 **Issue 5**

19090 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
19091 Threads Extension.

19092 Interactions with the SA\_NOCLDWAIT flag and SIGCHLD signal are further clarified.

19093 The values of *status* from `exit()` are better described.

19094 **Issue 6**

19095 Extensions beyond the ISO C standard are marked.

19096 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
19097 for typed memory.

19098 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 19099 • The `_Exit()` function is included.
- 19100 • The DESCRIPTION is updated.

19101 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

19102 References to the `wait3()` function are removed.

19103 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the  
19104 DESCRIPTION.

19105 **Issue 7**

19106 Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the  
19107 `exit()` function.

19108 Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of  
19109 streams and closing of temporary files.

19110 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and  
19111 Semaphores options is moved to the Base.

**\_longjmp()**19112 **NAME**

19113        \_longjmp, \_setjmp — non-local goto

19114 **SYNOPSIS**

```

19115 OB XSI #include <setjmp.h>
19116 void _longjmp(jmp_buf env, int val);
19117 int _setjmp(jmp_buf env);

```

19118 **DESCRIPTION**

19119        The *\_longjmp()* and *\_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,  
 19120        respectively, with the additional restriction that *\_longjmp()* and *\_setjmp()* shall not manipulate  
 19121        the signal mask.

19122        If *\_longjmp()* is called even though *env* was never initialized by a call to *\_setjmp()*, or when the  
 19123        last such call was in a function that has since returned, the results are undefined.

19124 **RETURN VALUE**19125        Refer to *longjmp()* and *setjmp()*.19126 **ERRORS**

19127        No errors are defined.

19128 **EXAMPLES**

19129        None.

19130 **APPLICATION USAGE**

19131        If *\_longjmp()* is executed and the environment in which *\_setjmp()* was executed no longer exists,  
 19132        errors can occur. The conditions under which the environment of the *\_setjmp()* no longer exists  
 19133        include exiting the function that contains the *\_setjmp()* call, and exiting an inner block with  
 19134        temporary storage. This condition might not be detectable, in which case the *\_longjmp()* occurs  
 19135        and, if the environment no longer exists, the contents of the temporary storage of an inner block  
 19136        are unpredictable. This condition might also cause unexpected process termination. If the  
 19137        function has returned, the results are undefined.

19138        Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *\_longjmp()* a pointer to a  
 19139        buffer not created by *\_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by  
 19140        *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user  
 19141        can cause all the problems listed above, and more.

19142        The *\_longjmp()* and *\_setjmp()* functions are included to support programs written to historical  
 19143        system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

19144 **RATIONALE**

19145        None.

19146 **FUTURE DIRECTIONS**19147        The *\_longjmp()* and *\_setjmp()* functions may be removed in a future version.19148 **SEE ALSO**19149        *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

19150        XBD &lt;setjmp.h&gt;

19151 **CHANGE HISTORY**

19152        First released in Issue 4, Version 2.

- 19153 **Issue 5**  
19154 Moved from X/OPEN UNIX extension to BASE.
- 19155 **Issue 7**  
19156 The *\_longjmp()* and *\_setjmp()* functions are marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**\_tolower()**19157 **NAME**

19158        \_toupper — transliterate uppercase characters to lowercase

19159 **SYNOPSIS**

```
19160 OB XSI #include <ctype.h>
19161         int _tolower(int c);
```

19162 **DESCRIPTION**

19163        The *\_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure  
 19164        that the argument *c* is an uppercase letter.

19165 **RETURN VALUE**

19166        Upon successful completion, *\_tolower()* shall return the lowercase letter corresponding to the  
 19167        argument passed.

19168 **ERRORS**

19169        No errors are defined.

19170 **EXAMPLES**

19171        None.

19172 **APPLICATION USAGE**19173        Applications should use the *tolower()* function instead of the obsolescent *\_tolower()* function.19174 **RATIONALE**

19175        None.

19176 **FUTURE DIRECTIONS**19177        The *\_tolower()* function may be removed in a future version.19178 **SEE ALSO**19179        *tolower()*, *isupper()*19180        XBD Chapter 7 (on page 135), *<ctype.h>*19181 **CHANGE HISTORY**

19182        First released in Issue 1. Derived from Issue 1 of the SVID.

19183 **Issue 6**

19184        The normative text is updated to avoid use of the term “must” for application requirements.

19185 **Issue 7**19186        The *\_tolower()* function is marked obsolescent.

19187 **NAME**

19188        \_toupper — transliterate lowercase characters to uppercase

19189 **SYNOPSIS**

```
19190 OB XSI #include <ctype.h>
19191         int _toupper(int c);
```

19192 **DESCRIPTION**

19193        The *\_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure  
 19194        that the argument *c* is a lowercase letter.

19195 **RETURN VALUE**

19196        Upon successful completion, *\_toupper()* shall return the uppercase letter corresponding to the  
 19197        argument passed.

19198 **ERRORS**

19199        No errors are defined.

19200 **EXAMPLES**

19201        None.

19202 **APPLICATION USAGE**

19203        Applications should use the *toupper()* function instead of the obsolescent *\_toupper()* function.

19204 **RATIONALE**

19205        None.

19206 **FUTURE DIRECTIONS**

19207        The *\_toupper()* function may be removed in a future version.

19208 **SEE ALSO**

19209        *islower()*, *toupper()*

19210        XBD Chapter 7 (on page 135), *<ctype.h>*

19211 **CHANGE HISTORY**

19212        First released in Issue 1. Derived from Issue 1 of the SVID.

19213 **Issue 6**

19214        The normative text is updated to avoid use of the term “must” for application requirements.

19215 **Issue 7**

19216        The *\_toupper()* function is marked obsolescent.

**a64l()**19217 **NAME**

19218 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

19219 **SYNOPSIS**

```
19220 xSI #include <stdlib.h>
19221 long a64l(const char *s);
19222 char *l64a(long value);
```

19223 **DESCRIPTION**

19224 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by  
 19225 which 32-bit integers can be represented by up to six characters; each character represents a digit  
 19226 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall  
 19227 be used for these operations.

19228 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],  
 19229 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

19230 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the  
 19231 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains  
 19232 more than six characters, *a64l()* shall use the first six. If the first six characters of the string  
 19233 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The  
 19234 *a64l()* function shall scan the character string from left to right with the least significant digit on  
 19235 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than  
 19236 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null  
 19237 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

19238 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding  
 19239 radix-64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

19240 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may  
 19241 overwrite the buffer.

19242 The *l64a()* function need not be thread-safe.

19243 **RETURN VALUE**

19244 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the  
 19245 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

19246 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall  
 19247 return a pointer to an empty string.

19248 **ERRORS**

19249 No errors are defined.

19250 **EXAMPLES**

19251 None.

19252 **APPLICATION USAGE**

19253 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

19254 **RATIONALE**

19255 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

19256 **FUTURE DIRECTIONS**

19257 None.

19258 **SEE ALSO**19259 [strtol\(\)](#)19260 XBD [<stdlib.h>](#)19261 XCU [uuencode](#)19262 **CHANGE HISTORY**

19263 First released in Issue 4, Version 2.

19264 **Issue 5**

19265 Moved from X/OPEN UNIX extension to BASE.

19266 Normative text previously in the APPLICATION USAGE section is moved to the  
19267 DESCRIPTION.19268 A note indicating that the *l64a()* function need not be reentrant is added to the DESCRIPTION.19269 **Issue 7**

19270 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**abort()**19271 **NAME**

19272 abort — generate an abnormal process abort

19273 **SYNOPSIS**

19274 #include &lt;stdlib.h&gt;

19275 void abort(void);

19276 **DESCRIPTION**

19277 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 19278 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19279 volume of POSIX.1-2008 defers to the ISO C standard.

19280 The *abort()* function shall cause abnormal process termination to occur, unless the signal  
 19281 SIGABRT is being caught and the signal handler does not return.

19282 CX The abnormal termination processing shall include the default actions defined for SIGABRT and  
 19283 may include an attempt to effect *fclose()* on all open streams.

19284 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the  
 19285 argument SIGABRT.

19286 CX The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort()* shall be that of a process  
 19287 terminated by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the  
 19288 SIGABRT signal.

19289 **RETURN VALUE**19290 The *abort()* function shall not return.19291 **ERRORS**

19292 No errors are defined.

19293 **EXAMPLES**

19294 None.

19295 **APPLICATION USAGE**

19296 Catching the signal is intended to provide the application developer with a portable means to  
 19297 abort processing, free from possible interference from any implementation-supplied functions.

19298 **RATIONALE**

19299 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since  
 19300 POSIX.1-2008 defers to the ISO C standard, this required a change to the DESCRIPTION from  
 19301 “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

19302 The revised wording permits some backwards-compatibility and avoids a potential deadlock  
 19303 situation.

19304 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded  
 19305 paragraph from the DESCRIPTION:

19306 “On XSI-conformant systems, in addition the abnormal termination processing shall include the  
 19307 effect of *fclose()* on message catalog descriptors.”

19308 There were several reasons to remove this paragraph:

- 19309 • No special processing of open message catalogs needs to be performed prior to abnormal  
 19310 process termination.
- 19311 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open  
 19312 streams is to flush output queued on the stream. Message catalogs in this context are read-  
 19313 only and, therefore, do not need to be flushed.

- 19314 • The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog  
 19315 descriptors are allowed, but not required to be implemented using a file descriptor, but  
 19316 there is no mention in POSIX.1-2008 of a message catalog descriptor using a standard I/O  
 19317 stream FILE object as would be expected by *fclose()*.

#### 19318 FUTURE DIRECTIONS

19319 None.

#### 19320 SEE ALSO

19321 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitid()*

19322 XBD <stdlib.h>

#### 19323 CHANGE HISTORY

19324 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 19325 Issue 6

19326 Extensions beyond the ISO C standard are marked.

19327 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

19328 The Open Group Base Resolution bwg2002-003 is applied.

19329 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the  
 19330 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

19331 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing “implementation-  
 19332 defined functions” to “implementation-supplied functions” in the APPLICATION USAGE  
 19333 section.

STANDARDSISO.COM : Click to view the full text of ISO/IEC/IEEE 9945:2009

**abs()**19334 **NAME**19335 `abs` — return an integer absolute value19336 **SYNOPSIS**19337 `#include <stdlib.h>`19338 `int abs(int i);`19339 **DESCRIPTION**19340 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
19341 conflict between the requirements described here and the ISO C standard is unintentional. This  
19342 volume of POSIX.1-2008 defers to the ISO C standard.19343 The `abs()` function shall compute the absolute value of its integer operand, *i*. If the result cannot  
19344 be represented, the behavior is undefined.19345 **RETURN VALUE**19346 The `abs()` function shall return the absolute value of its integer operand.19347 **ERRORS**

19348 No errors are defined.

19349 **EXAMPLES**

19350 None.

19351 **APPLICATION USAGE**19352 In two's-complement representation, the absolute value of the negative integer with largest  
19353 magnitude `{INT_MIN}` might not be representable.19354 **RATIONALE**

19355 None.

19356 **FUTURE DIRECTIONS**

19357 None.

19358 **SEE ALSO**19359 [\*`fabs\(\)`\*](#), [\*`labs\(\)`\*](#)19360 **XBD** `<stdlib.h>`19361 **CHANGE HISTORY**

19362 First released in Issue 1. Derived from Issue 1 of the SVID.

19363 **Issue 6**

19364 Extensions beyond the ISO C standard are marked.

19365 **NAME**

19366 accept — accept a new connection on a socket

19367 **SYNOPSIS**

19368 #include &lt;sys/socket.h&gt;

19369 int accept(int socket, struct sockaddr \*restrict address,  
19370 socklen\_t \*restrict address\_len);19371 **DESCRIPTION**19372 The *accept()* function shall extract the first connection on the queue of pending connections,  
19373 create a new socket with the same socket type protocol and address family as the specified  
19374 socket, and allocate a new file descriptor for that socket.19375 The *accept()* function takes the following arguments:19376 *socket* Specifies a socket that was created with *socket()*, has been bound to an address  
19377 with *bind()*, and has issued a successful call to *listen()*.19378 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of  
19379 the connecting socket shall be returned.19380 *address\_len* Points to a **socklen\_t** structure which on input specifies the length of the  
19381 supplied **sockaddr** structure, and on output specifies the length of the stored  
19382 address.19383 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored  
19384 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in  
19385 the object pointed to by *address\_len*.19386 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
19387 the stored address shall be truncated.19388 If the protocol permits connections by unbound clients, and the peer is not bound, then the  
19389 value stored in the object pointed to by *address* is unspecified.19390 If the listen queue is empty of connection requests and O\_NONBLOCK is not set on the file  
19391 descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is  
19392 empty of connection requests and O\_NONBLOCK is set on the file descriptor for the socket,  
19393 *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].19394 The accepted socket cannot itself accept more connections. The original socket remains open and  
19395 can accept more connections.19396 **RETURN VALUE**19397 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted  
19398 socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.19399 **ERRORS**19400 The *accept()* function shall fail if:

19401 [EAGAIN] or [EWOULDBLOCK]

19402 O\_NONBLOCK is set for the socket file descriptor and no connections are  
19403 present to be accepted.19404 [EBADF] The *socket* argument is not a valid file descriptor.

19405 [ECONNABORTED]

19406 A connection has been aborted.

**accept()**

|       |              |                                                                                                             |
|-------|--------------|-------------------------------------------------------------------------------------------------------------|
| 19407 | [EINTR]      | The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived. |
| 19408 |              |                                                                                                             |
| 19409 | [EINVAL]     | The <i>socket</i> is not accepting connections.                                                             |
| 19410 | [EMFILE]     | All file descriptors available to the process are currently open.                                           |
| 19411 | [ENFILE]     | The maximum number of file descriptors in the system are already open.                                      |
| 19412 | [ENOBUFS]    | No buffer space is available.                                                                               |
| 19413 | [ENOMEM]     | There was insufficient memory available to complete the operation.                                          |
| 19414 | [ENOTSOCK]   | The <i>socket</i> argument does not refer to a socket.                                                      |
| 19415 | [EOPNOTSUPP] | The <i>socket</i> type of the specified socket does not support accepting connections.                      |
| 19416 |              |                                                                                                             |

19417 The *accept()* function may fail if:

|       |                 |                                                                                                  |
|-------|-----------------|--------------------------------------------------------------------------------------------------|
| 19418 | OB XSR [EPROTO] | A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized. |
| 19419 |                 |                                                                                                  |

**EXAMPLES**

19420 None.

**APPLICATION USAGE**

19423 When a connection is available, *select()* indicates that the file descriptor for the socket is ready for reading.

**RATIONALE**

19426 None.

**FUTURE DIRECTIONS**

19428 None.

**SEE ALSO**

19430 *bind()*, *connect()*, *listen()*, *socket()*

19431 XBD <sys/socket.h>

**CHANGE HISTORY**

19433 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19434 The **restrict** keyword is added to the *accept()* prototype for alignment with the ISO/IEC 9899:1999 standard.

**Issue 7**

19437 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

19438 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS] and [ENOMEM] errors to become “shall fail” errors.

19440 Functionality relating to XSI STREAMS is marked obsolescent.

19441 **NAME**19442 `access, faccessat` — determine accessibility of a file relative to directory file descriptor19443 **SYNOPSIS**

```
19444 #include <unistd.h>
19445
19445 int access(const char *path, int amode);
19446 int faccessat(int fd, const char *path, int amode, int flag);
```

19447 **DESCRIPTION**

19448 The `access()` function shall check the file named by the pathname pointed to by the `path`  
 19449 argument for accessibility according to the bit pattern contained in `amode`, using the real user ID  
 19450 in place of the effective user ID and the real group ID in place of the effective group ID.

19451 The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked  
 19452 (R\_OK, W\_OK, X\_OK) or the existence test (F\_OK).

19453 If any access permissions are checked, each shall be checked individually, as described in XBD  
 19454 Section 4.4 (on page 108), except that where that description refers to execute permission for a  
 19455 process with appropriate privileges, an implementation may indicate success for X\_OK even if  
 19456 execute permission is not granted to any user.

19457 The `faccessat()` function shall be equivalent to the `access()` function, except in the case where `path`  
 19458 specifies a relative path. In this case the file whose accessibility is to be determined shall be  
 19459 located relative to the directory associated with the file descriptor `fd` instead of the current  
 19460 working directory. If the file descriptor was opened without O\_SEARCH, the function shall  
 19461 check whether directory searches are permitted using the current permissions of the directory  
 19462 underlying the file descriptor. If the file descriptor was opened with O\_SEARCH, the function  
 19463 shall not perform the check.

19464 If `faccessat()` is passed the special value AT\_FDCWD in the `fd` parameter, the current working  
 19465 directory is used and the behavior shall be identical to a call to `access()`.

19466 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 19467 in `<fcntl.h>`:

19468 AT\_EACCESS The checks for accessibility are performed using the effective user and group  
 19469 IDs instead of the real user and group ID as required in a call to `access()`.

19470 **RETURN VALUE**

19471 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 19472 return -1 and set `errno` to indicate the error.

19473 **ERRORS**

19474 These functions shall fail if:

19475 [EACCES] Permission bits of the file mode do not permit the requested access, or search  
 19476 permission is denied on a component of the path prefix.

19477 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 19478 argument.

19479 [ENAMETOOLONG]

19480 The length of a component of a pathname is longer than {NAME\_MAX}.

19481 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

19482 [ENOTDIR] A component of the path prefix is not a directory, or the `path` argument  
 19483 contains at least one non-`<slash>` character and ends with one or more trailing  
 19484 `<slash>` characters and the last pathname component names an existing file

**access()**

|       |                |                                                                                                                                                                      |
|-------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19485 |                | that is neither a directory nor a symbolic link to a directory.                                                                                                      |
| 19486 | [EROFS]        | Write access is requested for a file on a read-only file system.                                                                                                     |
| 19487 |                | The <i>faccessat()</i> function shall fail if:                                                                                                                       |
| 19488 | [EACCES]       | <i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.                                   |
| 19489 |                |                                                                                                                                                                      |
| 19490 | [EBADF]        | The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching. |
| 19491 |                |                                                                                                                                                                      |
| 19492 |                | These functions may fail if:                                                                                                                                         |
| 19493 | [EINVAL]       | The value of the <i>amode</i> argument is invalid.                                                                                                                   |
| 19494 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                                               |
| 19495 |                |                                                                                                                                                                      |
| 19496 | [ENAMETOOLONG] |                                                                                                                                                                      |
| 19497 |                | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.        |
| 19498 |                |                                                                                                                                                                      |
| 19499 |                |                                                                                                                                                                      |
| 19500 | [ETXTBSY]      | Write access is requested for a pure procedure (shared text) file that is being executed.                                                                            |
| 19501 |                |                                                                                                                                                                      |
| 19502 |                | The <i>faccessat()</i> function may fail if:                                                                                                                         |
| 19503 | [EINVAL]       | The value of the <i>flag</i> argument is not valid.                                                                                                                  |
| 19504 | [ENOTDIR]      | The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.                                |
| 19505 |                |                                                                                                                                                                      |

19506 **EXAMPLES**19507 **Testing for the Existence of a File**

19508 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
19509 #include <unistd.h>
19510 ...
19511 int result;
19512 const char*filename = "/tmp/myfile";
19513 result = access (filename, F_OK);
```

19514 **APPLICATION USAGE**

19515 Additional values of *amode* other than the set defined in the description may be valid; for  
19516 example, if a system has extended access controls.

19517 The use of the AT\_EACCESS value for *flag* enables functionality not available in *access()*.

19518 **RATIONALE**

19519 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*  
19520 function because:

- 19521 1. Historical implementations of *access()* do not test file access correctly when the process'  
19522 real user ID is superuser. In particular, they always return zero when testing execute  
19523 permissions without regard to whether the file is executable.

19524 2. The superuser has complete access to all files on a system. As a consequence, programs  
 19525 started by the superuser and switched to the effective user ID with lesser privileges  
 19526 cannot use *access()* to test their file access permissions.

19527 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of  
 19528 POSIX.1-2008 now allows *access()* to behave in the desired way because several implementations  
 19529 have corrected the problem. It was also argued that problem (2) is more easily solved by using  
 19530 *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather  
 19531 than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in  
 19532 this volume of POSIX.1-2008.

19533 The sentence concerning appropriate privileges and execute permission bits reflects the two  
 19534 possibilities implemented by historical implementations when checking superuser access for  
 19535 X\_OK.

19536 New implementations are discouraged from returning X\_OK unless at least one execution  
 19537 permission bit is set.

19538 The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in  
 19539 directories other than the current working directory without exposure to race conditions. Any  
 19540 part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified  
 19541 behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it  
 19542 can be guaranteed that the file tested for accessibility is located relative to the desired directory.

#### 19543 FUTURE DIRECTIONS

19544 None.

#### 19545 SEE ALSO

19546 *chmod()*, *fstatat()*

19547 XBD Section 4.4 (on page 108), [<fcntl.h>](#), [<unistd.h>](#)

#### 19548 CHANGE HISTORY

19549 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 19550 Issue 6

19551 The following new requirements on POSIX implementations derive from alignment with the  
 19552 Single UNIX Specification:

- 19553 • The [ELOOP] mandatory error condition is added.
- 19554 • A second [ENAMETOOLONG] is added as an optional error condition.
- 19555 • The [ETXTBSY] optional error condition is added.

19556 The following changes were made to align with the IEEE P1003.1a draft standard:

- 19557 • The [ELOOP] optional error condition is added.

#### 19558 Issue 7

19559 Austin Group Interpretations 1003.1-2001 #046 and #143 are applied.

19560 The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 19561 Set Part 2.

19562 Changes are made to allow a directory to be opened for searching.

19563 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 19564 pathname exists but is not a directory or a symbolic link to a directory.

**acos()**19565 **NAME**

19566           acos, acosf, acosl — arc cosine functions

19567 **SYNOPSIS**

```
19568       #include <math.h>
19569       double acos(double x);
19570       float  acosf(float x);
19571       long double acosl(long double x);
```

19572 **DESCRIPTION**

19573 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 19574 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19575 volume of POSIX.1-2008 defers to the ISO C standard.

19576       These functions shall compute the principal value of the arc cosine of their argument  $x$ . The  
 19577 value of  $x$  should be in the range  $[-1,1]$ .

19578       An application wishing to check for error situations should set *errno* to zero and call  
 19579 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 19580 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 19581 zero, an error has occurred.

19582 **RETURN VALUE**

19583       Upon successful completion, these functions shall return the arc cosine of  $x$ , in the range  $[0,\pi]$   
 19584 radians.

19585 **MX**       For finite values of  $x$  not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 19586 supported), or an implementation-defined value shall be returned.

19587 **MX**       If  $x$  is NaN, a NaN shall be returned.

19588       If  $x$  is  $+1$ ,  $+0$  shall be returned.

19589       If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 19590 defined value shall be returned.

19591 **ERRORS**

19592       These functions shall fail if:

19593 **MX**       Domain Error   The  $x$  argument is finite and is not in the range  $[-1,1]$ , or is  $\pm\text{Inf}$ .

19594       If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 19595 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 19596 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 19597 shall be raised.

19598 **EXAMPLES**

19599       None.

19600 **APPLICATION USAGE**

19601       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 19602 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19603 **RATIONALE**

19604       None.

19605 **FUTURE DIRECTIONS**

19606 None.

19607 **SEE ALSO**19608 *cos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

19609 XBD Section 4.19 (on page 116), &lt;math.h&gt;

19610 **CHANGE HISTORY**

19611 First released in Issue 1. Derived from Issue 1 of the SVID.

19612 **Issue 5**19613 The DESCRIPTION is updated to indicate how an application should check for an error. This  
19614 text was previously published in the APPLICATION USAGE section.19615 **Issue 6**19616 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.19617 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
19618 revised to align with the ISO/IEC 9899:1999 standard.19619 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
19620 marked.

**acosh()**19621 **NAME**19622 `acosh, acoshf, acoshl` — inverse hyperbolic cosine functions19623 **SYNOPSIS**

```
19624 #include <math.h>
19625 double acosh(double x);
19626 float acoshf(float x);
19627 long double acoshl(long double x);
```

19628 **DESCRIPTION**

19629 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 19630 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19631 volume of POSIX.1-2008 defers to the ISO C standard.

19632 These functions shall compute the inverse hyperbolic cosine of their argument  $x$ .

19633 An application wishing to check for error situations should set *errno* to zero and call  
 19634 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 19635 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 19636 zero, an error has occurred.

19637 **RETURN VALUE**

19638 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their  
 19639 argument.

19640 **MX** For finite values of  $x < 1$ , a domain error shall occur, and either a NaN (if supported), or an  
 19641 implementation-defined value shall be returned.

19642 **MX** If  $x$  is NaN, a NaN shall be returned.

19643 If  $x$  is +1, +0 shall be returned.

19644 If  $x$  is +Inf, +Inf shall be returned.

19645 If  $x$  is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 19646 defined value shall be returned.

19647 **ERRORS**

19648 These functions shall fail if:

19649 **MX** Domain Error The  $x$  argument is finite and less than +1.0, or is -Inf.

19650 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 19651 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling* &  
 19652 MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 19653 shall be raised.

19654 **EXAMPLES**

19655 None.

19656 **APPLICATION USAGE**

19657 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 19658 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19659 **RATIONALE**

19660 None.

19661 **FUTURE DIRECTIONS**

19662 None.

19663 **SEE ALSO**19664 *cosh()*, *feclearexcept()*, *fetestexcept()*19665 XBD Section 4.19 (on page 116), `<math.h>`19666 **CHANGE HISTORY**

19667 First released in Issue 4, Version 2.

19668 **Issue 5**

19669 Moved from X/OPEN UNIX extension to BASE.

19670 **Issue 6**19671 The *acosh()* function is no longer marked as an extension.19672 The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

19674 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

19676 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

**acosl()***System Interfaces*19678 **NAME**

19679           acosl — arc cosine functions

19680 **SYNOPSIS**

19681           #include &lt;math.h&gt;

19682           long double acosl(long double x);

19683 **DESCRIPTION**19684           Refer to *acos()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

19685 **NAME**

19686 aio\_cancel — cancel an asynchronous I/O request

19687 **SYNOPSIS**

19688 #include &lt;aio.h&gt;

19689 int aio\_cancel(int *fildev*, struct aiocb \**aiocbp*);19690 **DESCRIPTION**

19691 The *aio\_cancel()* function shall attempt to cancel one or more asynchronous I/O requests  
 19692 currently outstanding against file descriptor *fildev*. The *aiocbp* argument points to the  
 19693 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then  
 19694 all outstanding cancelable asynchronous I/O requests against *fildev* shall be canceled.

19695 Normal asynchronous notification shall occur for asynchronous I/O operations that are  
 19696 successfully canceled. If there are requests that cannot be canceled, then the normal  
 19697 asynchronous completion process shall take place for those requests when they are completed.

19698 For requested operations that are successfully canceled, the associated error status shall be set to  
 19699 [ECANCELED] and the return status shall be -1. For requested operations that are not  
 19700 successfully canceled, the *aiocbp* shall not be modified by *aio\_cancel()*.

19701 If *aiocbp* is not NULL, then if *fildev* does not have the same value as the file descriptor with which  
 19702 the asynchronous operation was initiated, unspecified results occur.

19703 Which operations are cancelable is implementation-defined.

19704 **RETURN VALUE**

19705 The *aio\_cancel()* function shall return the value AIO\_CANCELED if the requested operation(s)  
 19706 were canceled. The value AIO\_NOTCANCELED shall be returned if at least one of the  
 19707 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the  
 19708 other operations, if any, referenced in the call to *aio\_cancel()* is not indicated by the return value  
 19709 of *aio\_cancel()*. The application may determine the state of affairs for these operations by using  
 19710 *aio\_error()*. The value AIO\_ALLDONE is returned if all of the operations have already  
 19711 completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

19712 **ERRORS**

19713 The *aio\_cancel()* function shall fail if:

19714 [EBADF] The *fildev* argument is not a valid file descriptor.

19715 **EXAMPLES**

19716 None.

19717 **APPLICATION USAGE**

19718 None.

19719 **RATIONALE**

19720 None.

19721 **FUTURE DIRECTIONS**

19722 None.

19723 **SEE ALSO**

19724 *aio\_read()*, *aio\_write()*

19725 XBD <aio.h>

**aio\_cancel()**

System Interfaces

19726 **CHANGE HISTORY**

19727 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19728 **Issue 6**19729 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
19730 implementation does not support the Asynchronous Input and Output option.

19731 The APPLICATION USAGE section is added.

19732 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words “to the  
19733 calling process” in the RETURN VALUE section. The term was unnecessary and precluded  
19734 threads.19735 **Issue 7**19736 The *aio\_cancel()* function is moved from the Asynchronous Input and Output option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

19737 **NAME**

19738 aio\_error — retrieve errors status for an asynchronous I/O operation

19739 **SYNOPSIS**

19740 #include &lt;aio.h&gt;

19741 int aio\_error(const struct aiocb \*aiocbp);

19742 **DESCRIPTION**

19743 The *aio\_error()* function shall return the error status associated with the **aiocb** structure  
 19744 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the  
 19745 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*  
 19746 operation. If the operation has not yet completed, then the error status shall be equal to  
 19747 [EINPROGRESS].

19748 If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been  
 19749 scheduled, the results are undefined.

19750 **RETURN VALUE**

19751 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the  
 19752 asynchronous operation has completed unsuccessfully, then the error status, as described for  
 19753 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has  
 19754 not yet completed, then [EINPROGRESS] shall be returned.

19755 If the *aio\_error()* function fails, it shall return -1 and set *errno* to indicate the error.

19756 **ERRORS**19757 The *aio\_error()* function may fail if:

19758 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
 19759 return status has not yet been retrieved.

19760 **EXAMPLES**

19761 None.

19762 **APPLICATION USAGE**

19763 None.

19764 **RATIONALE**

19765 None.

19766 **FUTURE DIRECTIONS**

19767 None.

19768 **SEE ALSO**

19769 *aio\_cancel()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
 19770 *lseek()*, *read()*

19771 XBD &lt;aio.h&gt;

19772 **CHANGE HISTORY**

19773 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19774 **Issue 6**

19775 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 19776 implementation does not support the Asynchronous Input and Output option.

19777 The APPLICATION USAGE section is added.

**aio\_error()**19778 **Issue 7**

19779 Austin Group Interpretation 1003.1-2001 #045 is applied.

19780 SD5-XSH-ERN-148 is applied.

19781 The *aio\_error()* function is moved from the Asynchronous Input and Output option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

19782 **NAME**

19783 aio\_fsync — asynchronous file synchronization

19784 **SYNOPSIS**

19785 #include &lt;aio.h&gt;

19786 int aio\_fsync(int *op*, struct aiocb \**aiocbp*);19787 **DESCRIPTION**

19788 The *aio\_fsync()* function shall asynchronously force all I/O operations associated with the file  
 19789 indicated by the file descriptor *aio\_fildes* member of the **aiocb** structure referenced by the *aiocbp*  
 19790 argument and queued at the time of the call to *aio\_fsync()* to the synchronized I/O completion  
 19791 state. The function call shall return when the synchronization request has been initiated or  
 19792 queued to the file or device (even when the data cannot be synchronized immediately).

19793 If *op* is O\_DSYNC, all currently queued I/O operations shall be completed as if by a call to  
 19794 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O\_SYNC,  
 19795 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as  
 19796 defined for synchronized I/O file integrity completion. If the *aio\_fsync()* function fails, or if the  
 19797 operation queued by *aio\_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O  
 19798 operations are not guaranteed to have been completed.

19799 If *aio\_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to  
 19800 *aio\_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of  
 19801 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized  
 19802 fashion.

19803 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used  
 19804 as an argument to *aio\_error()* and *aio\_return()* in order to determine the error status and return  
 19805 status, respectively, of the asynchronous operation while it is proceeding. When the request is  
 19806 queued, the error status for the operation is [EINPROGRESS]. When all data has been  
 19807 successfully transferred, the error status shall be reset to reflect the success or failure of the  
 19808 operation. If the operation does not complete successfully, the error status for the operation shall  
 19809 be set to indicate the error. The *aio\_sigevent* member determines the asynchronous notification to  
 19810 occur as specified in Section 2.4.1 (on page 484) when all operations have achieved synchronized  
 19811 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the  
 19812 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 19813 completion, then the behavior is undefined.

19814 If the *aio\_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to  
 19815 have been successfully transferred.

19816 **RETURN VALUE**

19817 The *aio\_fsync()* function shall return the value 0 if the I/O operation is successfully queued;  
 19818 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

19819 **ERRORS**

19820 The *aio\_fsync()* function shall fail if:

- |       |          |                                                                                                                                                      |
|-------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19821 | [EAGAIN] | The requested asynchronous operation was not queued due to temporary resource limitations.                                                           |
| 19822 |          |                                                                                                                                                      |
| 19823 | [EBADF]  | The <i>aio_fildes</i> member of the <b>aiocb</b> structure referenced by the <i>aiocbp</i> argument is not a valid file descriptor open for writing. |
| 19824 |          |                                                                                                                                                      |
| 19825 | [EINVAL] | This implementation does not support synchronized I/O for this file.                                                                                 |

**aio\_fsync()**

System Interfaces

- 19826 [EINVAL] A value of *op* other than O\_DSYNC or O\_SYNC was specified.
- 19827 In the event that any of the queued I/O operations fail, *aio\_fsync()* shall return the error  
19828 condition defined for *read()* and *write()*. The error is returned in the error status for the  
19829 asynchronous *fsync()* operation, which can be retrieved using *aio\_error()*.
- 19830 **EXAMPLES**
- 19831 None.
- 19832 **APPLICATION USAGE**
- 19833 None.
- 19834 **RATIONALE**
- 19835 None.
- 19836 **FUTURE DIRECTIONS**
- 19837 None.
- 19838 **SEE ALSO**
- 19839 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*
- 19840 XBD <[aio.h](#)>
- 19841 **CHANGE HISTORY**
- 19842 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 19843 **Issue 6**
- 19844 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
19845 implementation does not support the Asynchronous Input and Output option.
- 19846 The APPLICATION USAGE section is added.
- 19847 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the  
19848 calling process” in the RETURN VALUE section. The term was unnecessary and precluded  
19849 threads.
- 19850 **Issue 7**
- 19851 The *aio\_fsync()* function is moved from the Asynchronous Input and Output option to the Base.

19852 **NAME**

19853 aio\_read — asynchronous read from a file

19854 **SYNOPSIS**

19855 #include &lt;aio.h&gt;

19856 int aio\_read(struct aiocb \*aiocbp);

19857 **DESCRIPTION**

19858 The *aio\_read()* function shall read *aiocbp->aio\_nbytes* from the file associated with  
 19859 *aiocbp->aio\_fildes* into the buffer pointed to by *aiocbp->aio\_buf*. The function call shall return  
 19860 when the read request has been initiated or queued to the file or device (even when the data  
 19861 cannot be delivered immediately).

19862 **PIO** If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted  
 19863 at a priority equal to a base scheduling priority minus *aiocbp->aio\_reqprio*. If Thread Execution  
 19864 Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 19865 **PIO TPS** otherwise, the base scheduling priority is that of the calling thread.

19866 The *aiocbp* value may be used as an argument to *aio\_error()* and *aio\_return()* in order to  
 19867 determine the error status and return status, respectively, of the asynchronous operation while it  
 19868 is proceeding. If an error condition is encountered during queuing, the function call shall return  
 19869 without having initiated or queued the request. The requested operation takes place at the  
 19870 absolute position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to  
 19871 the operation with an *offset* equal to *aio\_offset* and a *whence* equal to *SEEK\_SET*. After a  
 19872 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file  
 19873 is unspecified.

19874 The *aio\_sigevent* member specifies the notification which occurs when the request is completed.

19875 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_read()*.

19876 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or  
 19877 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 19878 completion, then the behavior is undefined.

19879 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

19880 **SIO** If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this  
 19881 function shall be according to the definitions of synchronized I/O data integrity completion and  
 19882 synchronized I/O file integrity completion.

19883 For any system action that changes the process memory space while an asynchronous I/O is  
 19884 outstanding to the address range being changed, the result of that action is undefined.

19885 For regular files, no data transfer shall occur past the offset maximum established in the open  
 19886 file description associated with *aiocbp->aio\_fildes*.

19887 **RETURN VALUE**

19888 The *aio\_read()* function shall return the value zero if the I/O operation is successfully queued;  
 19889 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

19890 **ERRORS**

19891 The *aio\_read()* function shall fail if:

19892 [EAGAIN] The requested asynchronous I/O operation was not queued due to system  
 19893 resource limitations.

19894 Each of the following conditions may be detected synchronously at the time of the call to  
 19895 *aio\_read()*, or asynchronously. If any of the conditions below are detected synchronously, the

**aio\_read()**

- 19896 *aio\_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the  
 19897 conditions below are detected asynchronously, the return status of the asynchronous operation  
 19898 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.
- 19899 [EBADF] The *aioctx->aio\_fildes* argument is not a valid file descriptor open for reading.
- 19900 [EINVAL] The file offset value implied by *aioctx->aio\_offset* would be invalid,  
 19901 PIO *aioctx->aio\_reqprio* is not a valid value, or *aioctx->aio\_nbytes* is an invalid  
 19902 value.
- 19903 In the case that the *aio\_read()* successfully queues the I/O operation but the operation is  
 19904 subsequently canceled or encounters an error, the return status of the asynchronous operation is  
 19905 one of the values normally returned by the *read()* function call. In addition, the error status of  
 19906 the asynchronous operation is set to one of the error statuses normally set by the *read()* function  
 19907 call, or one of the following values:
- 19908 [EBADF] The *aioctx->aio\_fildes* argument is not a valid file descriptor open for reading.
- 19909 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 19910 *aio\_cancel()* request.
- 19911 [EINVAL] The file offset value implied by *aioctx->aio\_offset* would be invalid.
- 19912 The following condition may be detected synchronously or asynchronously:
- 19913 [EOVERFLOW] The file is a regular file, *aioctx->aio\_nbytes* is greater than 0, and the starting  
 19914 offset in *aioctx->aio\_offset* is before the end-of-file and is at or beyond the  
 19915 offset maximum in the open file description associated with *aioctx->aio\_fildes*.

**EXAMPLES**

19916 None.

**APPLICATION USAGE**

19919 None.

**RATIONALE**

19921 None.

**FUTURE DIRECTIONS**

19923 None.

**SEE ALSO**

19925 *aio\_cancel()*, *aio\_error()*, *lio\_listio()*, *aio\_return()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,  
 19926 *read()*

19927 XBD <[aio.h](#)>

**CHANGE HISTORY**

19929 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19930 Large File Summit extensions are added.

**Issue 6**

19932 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 19933 implementation does not support the Asynchronous Input and Output option.

19934 The APPLICATION USAGE section is added.

- 19935 The following new requirements on POSIX implementations derive from alignment with the  
19936 Single UNIX Specification:
- 19937 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
19938 file description. This change is to support large files.
  - 19939 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
19940 large files.
- 19941 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the  
19942 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the  
19943 words “to the calling process” in the RETURN VALUE section.
- 19944 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL]  
19945 error, so that detection of an [EINVAL] error for an invalid value of *aiochp*→*aio\_reqprio* is only  
19946 required if the Prioritized Input and Output option is supported.
- 19947 **Issue 7**
- 19948 Austin Group Interpretation 1003.1-2001 #082 is applied.
- 19949 The *aio\_read()* function is moved from the Asynchronous Input and Output option to the Base.

**aio\_return()**19950 **NAME**

19951 aio\_return — retrieve return status of an asynchronous I/O operation

19952 **SYNOPSIS**

19953 #include &lt;aio.h&gt;

19954 ssize\_t aio\_return(struct aiocb \*aiocbp);

19955 **DESCRIPTION**

19956 The *aio\_return()* function shall return the return status associated with the **aiocb** structure  
 19957 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the  
 19958 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the  
 19959 error status for the operation is equal to [EINPROGRESS], then the return status for the  
 19960 operation is undefined. The *aio\_return()* function may be called exactly once to retrieve the  
 19961 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in  
 19962 a call to *aio\_return()* or *aio\_error()*, an error may be returned. When the **aiocb** structure referred  
 19963 to by *aiocbp* is used to submit another asynchronous operation, then *aio\_return()* may be  
 19964 successfully used to retrieve the return status of that operation.

19965 **RETURN VALUE**

19966 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,  
 19967 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,  
 19968 the results of *aio\_return()* are undefined.

19969 If the *aio\_return()* function fails, it shall return -1 and set *errno* to indicate the error.

19970 **ERRORS**19971 The *aio\_return()* function may fail if:

19972 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
 19973 return status has not yet been retrieved.

19974 **EXAMPLES**

19975 None.

19976 **APPLICATION USAGE**

19977 None.

19978 **RATIONALE**

19979 None.

19980 **FUTURE DIRECTIONS**

19981 None.

19982 **SEE ALSO**

19983 *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
 19984 *lseek()*, *read()*

19985 XBD &lt;aio.h&gt;

19986 **CHANGE HISTORY**

19987 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19988 **Issue 6**

19989 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 19990 implementation does not support the Asynchronous Input and Output option.

19991 The APPLICATION USAGE section is added.

19992 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

19993 **Issue 7**

19994 SD5-XSH-ERN-148 is applied.

19995 The *aio\_return()* function is moved from the Asynchronous Input and Output option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**aio\_suspend()**19996 **NAME**

19997 aio\_suspend — wait for an asynchronous I/O request

19998 **SYNOPSIS**

19999 #include &lt;aio.h&gt;

20000 int aio\_suspend(const struct aiocb \*const list[], int nent,  
20001 const struct timespec \*timeout);20002 **DESCRIPTION**

20003 The *aio\_suspend()* function shall suspend the calling thread until at least one of the asynchronous  
 20004 I/O operations referenced by the *list* argument has completed, until a signal interrupts the  
 20005 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any  
 20006 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,  
 20007 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the  
 20008 function shall return without suspending the calling thread. The *list* argument is an array of  
 20009 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of  
 20010 elements in the array. Each **aiocb** structure pointed to has been used in initiating an  
 20011 asynchronous I/O request via *aio\_read()*, *aio\_write()*, or *lio\_listio()*. This array may contain null  
 20012 pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have  
 20013 not been used in submitting asynchronous I/O, the effect is undefined.

20014 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of  
 20015 the I/O operations referenced by *list* are completed, then *aio\_suspend()* shall return with an error.

20016 MON If the Monotonic Clock option is supported, the clock that shall be used to measure this time  
 20017 interval shall be the CLOCK\_MONOTONIC clock.

20018 **RETURN VALUE**

20019 If the *aio\_suspend()* function returns after one or more asynchronous I/O operations have  
 20020 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and  
 20021 set *errno* to indicate the error.

20022 The application may determine which asynchronous I/O completed by scanning the associated  
 20023 error and return status using *aio\_error()* and *aio\_return()*, respectively.

20024 **ERRORS**

20025 The *aio\_suspend()* function shall fail if:

20026 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the  
 20027 time interval indicated by *timeout*.

20028 [EINTR] A signal interrupted the *aio\_suspend()* function. Note that, since each  
 20029 asynchronous I/O operation may possibly provoke a signal when it  
 20030 completes, this error return may be caused by the completion of one (or more)  
 20031 of the very I/O operations being awaited.

20032 **EXAMPLES**

20033 None.

20034 **APPLICATION USAGE**

20035 None.

20036 **RATIONALE**

20037 None.

20038 **FUTURE DIRECTIONS**

20039 None.

20040 **SEE ALSO**20041 *aio\_read()*, *aio\_write()*, *lio\_listio()*20042 XBD <*aio.h*>20043 **CHANGE HISTORY**

20044 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20045 **Issue 6**20046 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
20047 implementation does not support the Asynchronous Input and Output option.

20048 The APPLICATION USAGE section is added.

20049 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
20050 CLOCK\_MONOTONIC clock, if supported, is used.20051 **Issue 7**20052 The *aio\_suspend()* function is moved from the Asynchronous Input and Output option to the  
20053 Base.

**aio\_write()**20054 **NAME**

20055 aio\_write — asynchronous write to a file

20056 **SYNOPSIS**

20057 #include &lt;aio.h&gt;

20058 int aio\_write(struct aiocb \*aiocbp);

20059 **DESCRIPTION**

20060 The *aio\_write()* function shall write *aiocbp->aio\_nbytes* to the file associated with  
 20061 *aiocbp->aio\_fildes* from the buffer pointed to by *aiocbp->aio\_buf*. The function shall return when  
 20062 the write request has been initiated or, at a minimum, queued to the file or device.

20063 **PIO** If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted  
 20064 at a priority equal to a base scheduling priority minus *aiocbp->aio\_reqprio*. If Thread Execution  
 20065 Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 20066 **PIO TPS** otherwise, the base scheduling priority is that of the calling thread.

20067 The *aiocbp* argument may be used as an argument to *aio\_error()* and *aio\_return()* in order to  
 20068 determine the error status and return status, respectively, of the asynchronous operation while it  
 20069 is proceeding.

20070 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or  
 20071 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 20072 completion, then the behavior is undefined.

20073 If **O\_APPEND** is not set for the file descriptor *aio\_fildes*, then the requested operation shall take  
 20074 place at the absolute position in the file as given by *aio\_offset*, as if *lseek()* were called  
 20075 immediately prior to the operation with an *offset* equal to *aio\_offset* and a *whence* equal to  
 20076 **SEEK\_SET**. If **O\_APPEND** is set for the file descriptor, write operations append to the file in the  
 20077 same order as the calls were made. After a successful call to enqueue an asynchronous I/O  
 20078 operation, the value of the file offset for the file is unspecified.

20079 The *aio\_sigevent* member specifies the notification which occurs when the request is completed.

20080 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_write()*.

20081 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

20082 **SIO** If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this  
 20083 function shall be according to the definitions of synchronized I/O data integrity completion, and  
 20084 synchronized I/O file integrity completion.

20085 For any system action that changes the process memory space while an asynchronous I/O is  
 20086 outstanding to the address range being changed, the result of that action is undefined.

20087 For regular files, no data transfer shall occur past the offset maximum established in the open  
 20088 file description associated with *aiocbp->aio\_fildes*.

20089 **RETURN VALUE**

20090 The *aio\_write()* function shall return the value zero if the I/O operation is successfully queued;  
 20091 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

20092 **ERRORS**

20093 The *aio\_write()* function shall fail if:

20094 **[EAGAIN]** The requested asynchronous I/O operation was not queued due to system  
 20095 resource limitations.

20096 Each of the following conditions may be detected synchronously at the time of the call to  
 20097 *aio\_write()*, or asynchronously. If any of the conditions below are detected synchronously, the

20098 *aio\_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the  
 20099 conditions below are detected asynchronously, the return status of the asynchronous operation  
 20100 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding  
 20101 value.

20102 [EBADF] The *aiocbp*→*aio\_fildes* argument is not a valid file descriptor open for writing.

20103 [EINVAL] The file offset value implied by *aiocbp*→*aio\_offset* would be invalid,  
 20104 PIO *aiocbp*→*aio\_reqprio* is not a valid value, or *aiocbp*→*aio\_nbytes* is an invalid  
 20105 value.

20106 In the case that the *aio\_write()* successfully queues the I/O operation, the return status of the  
 20107 asynchronous operation shall be one of the values normally returned by the *write()* function call.  
 20108 If the operation is successfully queued but is subsequently canceled or encounters an error, the  
 20109 error status for the asynchronous operation contains one of the values normally set by the  
 20110 *write()* function call, or one of the following:

20111 [EBADF] The *aiocbp*→*aio\_fildes* argument is not a valid file descriptor open for writing.

20112 [EINVAL] The file offset value implied by *aiocbp*→*aio\_offset* would be invalid.

20113 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 20114 *aio\_cancel()* request.

20115 The following condition may be detected synchronously or asynchronously:

20116 [EFBIG] The file is a regular file, *aiocbp*→*aio\_nbytes* is greater than 0, and the starting  
 20117 offset in *aiocbp*→*aio\_offset* is at or beyond the offset maximum in the open file  
 20118 description associated with *aiocbp*→*aio\_fildes*.

#### 20119 EXAMPLES

20120 None.

#### 20121 APPLICATION USAGE

20122 None.

#### 20123 RATIONALE

20124 None.

#### 20125 FUTURE DIRECTIONS

20126 None.

#### 20127 SEE ALSO

20128 *aio\_cancel()*, *aio\_error()*, *aio\_read()*, *aio\_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*, *lseek()*,  
 20129 *write()*

20130 XBD <[aio.h](#)>

#### 20131 CHANGE HISTORY

20132 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20133 Large File Summit extensions are added.

#### 20134 Issue 6

20135 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 20136 implementation does not support the Asynchronous Input and Output option.

20137 The APPLICATION USAGE section is added.

20138 The following new requirements on POSIX implementations derive from alignment with the  
 20139 Single UNIX Specification:

**aio\_write()**

- 20140
- 20141
- 20142
- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp*→*aio\_fildes*.
- 20143
- The [EFBIG] error is added as part of the large file support extensions.
- 20144
- 20145
- 20146
- IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words “to the calling process” in the RETURN VALUE section.
- 20147
- 20148
- 20149
- IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp*→*aio\_reqprio* is only required if the Prioritized Input and Output option is supported.
- 20150 **Issue 7**
- 20151 Austin Group Interpretation 1003.1-2001 #082 is applied.
- 20152 The *aio\_write()* function is moved from the Asynchronous Input and Output option to the Base.

20153 **NAME**

20154 alarm — schedule an alarm signal

20155 **SYNOPSIS**

20156 #include &lt;unistd.h&gt;

20157 unsigned alarm(unsigned seconds);

20158 **DESCRIPTION**

20159 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after  
 20160 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays  
 20161 may prevent the process from handling the signal as soon as it is generated.

20162 If *seconds* is 0, a pending alarm request, if any, is canceled.

20163 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.  
 20164 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time  
 20165 at which the SIGALRM signal is generated.

20166 XSI Interactions between *alarm()* and *setitimer()* are unspecified.

20167 **RETURN VALUE**

20168 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value  
 20169 that is the number of seconds until the previous request would have generated a SIGALRM  
 20170 signal. Otherwise, *alarm()* shall return 0.

20171 **ERRORS**

20172 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

20173 **EXAMPLES**

20174 None.

20175 **APPLICATION USAGE**

20176 The *fork()* function clears pending alarms in the child process. A new process image created by  
 20177 one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

20178 Application developers should note that the type of the argument *seconds* and the return value of  
 20179 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces  
 20180 Application cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX},  
 20181 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting  
 20182 its portability. A different type was considered, but historical implementations, including those  
 20183 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

20184 Application developers should be aware of possible interactions when the same process uses  
 20185 both the *alarm()* and *sleep()* functions.

20186 **RATIONALE**

20187 Many historical implementations (including Version 7 and System V) allow an alarm to occur up  
 20188 to a second early. Other implementations allow alarms up to half a second or one clock tick  
 20189 early or do not allow them to occur early at all. The latter is considered most appropriate, since it  
 20190 gives the most predictable behavior, especially since the signal can always be delayed for an  
 20191 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument  
 20192 as the minimum amount of time they wish to have elapse before the signal.

20193 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time  
 20194 as common English usage, and has nothing to do with “realtime operating systems”. It is in  
 20195 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

20196 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are  
 20197 silently rounded down to an implementation-specific maximum value. This maximum is large

**alarm()**

- 20198 enough (to the order of several months) that the effect is not noticeable.
- 20199 There were two possible choices for alarm generation in multi-threaded applications: generation  
20200 for the calling thread or generation for the process. The first option would not have been  
20201 particularly useful since the alarm state is maintained on a per-process basis and the alarm that  
20202 is established by the last invocation of *alarm()* is the only one that would be active.
- 20203 Furthermore, allowing generation of an asynchronous signal for a thread would have  
20204 introduced an exception to the overall signal model. This requires a compelling reason in order  
20205 to be justified.
- 20206 **FUTURE DIRECTIONS**
- 20207 None.
- 20208 **SEE ALSO**
- 20209 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*
- 20210 XBD [<signal.h>](#), [<unistd.h>](#)
- 20211 **CHANGE HISTORY**
- 20212 First released in Issue 1. Derived from Issue 1 of the SVID.
- 20213 **Issue 6**
- 20214 The following new requirements on POSIX implementations derive from alignment with the  
20215 Single UNIX Specification:
- 20216 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*,  
20217 and *usleep()* functions are unspecified.
- 20218 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing “an  
20219 implementation-defined maximum value” with “an implementation-specific maximum value”  
20220 in the RATIONALE.

20221 **NAME**

20222           alphasort, scandir — scan a directory

20223 **SYNOPSIS**

20224           #include &lt;dirent.h&gt;

```

20225           int alphasort(const struct dirent **d1, const struct dirent **d2);
20226           int scandir(const char *dir, struct dirent ***namelist,
20227                       int (*sel)(const struct dirent *),
20228                       int (*compar)(const struct dirent **, const struct dirent **));

```

20229 **DESCRIPTION**

20230       The *alphasort()* function can be used as the comparison function for the *scandir()* function to sort  
 20231       the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the  
 20232       *strcoll()* function on the *d\_name* element of the **dirent** structures passed as the two parameters. If  
 20233       the *strcoll()* function fails, the return value of *alphasort()* is unspecified.

20234       The *alphasort()* function shall not change the setting of *errno* if successful. Since no return value  
 20235       is reserved to indicate an error, an application wishing to check for error situations should set  
 20236       *errno* to 0, then call *alphasort()*, then check *errno*.

20237       The *scandir()* function shall scan the directory *dir*, calling the function referenced by *sel* on each  
 20238       directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored  
 20239       in strings allocated as if by a call to *malloc()*, and sorted as if by a call to *qsort()* with the  
 20240       comparison function *compar*, except that *compar* need not provide total ordering. The strings are  
 20241       collected in array *namelist* which shall be allocated as if by a call to *malloc()*. If *sel* is a null  
 20242       pointer, all entries shall be selected. If the comparison function *compar* does not provide total  
 20243       ordering, the order in which the directory entries are stored is unspecified.

20244 **RETURN VALUE**

20245       Upon successful completion, the *alphasort()* function shall return an integer greater than, equal  
 20246       to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is  
 20247       lexically greater than, equal to, or less than the directory pointed to by *d2* when both are  
 20248       interpreted as appropriate to the current locale. There is no return value reserved to indicate an  
 20249       error.

20250       Upon successful completion, the *scandir()* function shall return the number of entries in the  
 20251       array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir()*  
 20252       function shall return -1.

20253 **ERRORS**20254       The *scandir()* function shall fail if:

20255       [EACCES]       Search permission is denied for the component of the path prefix of *dir* or read  
 20256       permission is denied for *dir*.

20257       [ELOOP]        A loop exists in symbolic links encountered during resolution of the *dir*  
 20258       argument.

20259       [ENAMETOOLONG]   The length of a component of a pathname is longer than {NAME\_MAX}.

20261       [ENOENT]        A component of *dir* does not name an existing directory or *dir* is an empty  
 20262       string.

20263       [ENOMEM]        Insufficient storage space is available.

**alphasort()**

- 20264 [ENOTDIR] A component of *dir* is not a directory.
- 20265 The *scandir()* function may fail if:
- 20266 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
20267 resolution of the *dir* argument.
- 20268 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.
- 20269 [ENAMETOOLONG]
- 20270 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
20271 symbolic link produced an intermediate result with a length that exceeds  
20272 {PATH\_MAX}.
- 20273 [ENFILE] Too many files are currently open in the system.

**EXAMPLES**

20274 An example to print the files in the current directory:

```

20276 #include <dirent.h>
20277 #include <stdio.h>
20278 #include <stdlib.h>
20279 ...
20280 struct dirent **namelist;
20281 int i,n;

20282     n = scandir(".", &namelist, 0, alphasort);
20283     if (n < 0)
20284         perror("scandir");
20285     else {
20286         for (i = 0; i < n; i++) {
20287             printf("%s\n", namelist[i]->d_name);
20288             free(namelist[i]);
20289         }
20290     }
20291     free(namelist);
20292     ...

```

**APPLICATION USAGE**

20293 If *dir* contains filenames that contain characters outside the domain of the collating sequence of  
20294 the current locale, the *alphasort()* function need not provide a total ordering.

20296 The *scandir()* function may allocate dynamic storage during its operation. If *scandir()* is forcibly  
20297 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *sel*  
20298 or *compar*, or by an interrupt routine, *scandir()* does not have a chance to free that storage, so it  
20299 remains permanently allocated. A safe way to handle interrupts is to store the fact that an  
20300 interrupt has occurred, then wait until *scandir()* returns to act on the interrupt.

20301 For functions that allocate memory as if by *malloc()*, the application should release such memory  
20302 when it is no longer required by a call to *free()*. For *scandir()*, this is *namelist* (including all of the  
20303 individual strings in *namelist*).

**RATIONALE**

20304 None.

20306 **FUTURE DIRECTIONS**

20307 None.

20308 **SEE ALSO**20309 *qsort()*, *strcoll()*20310 XBD <**dirent.h**>20311 **CHANGE HISTORY**

20312 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**asctime()**20313 **NAME**20314 `asctime, asctime_r` — convert date and time to a string20315 **SYNOPSIS**

```
20316 OB #include <time.h>
20317 char *asctime(const struct tm *timeptr);
20318 OB CX char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

20319 **DESCRIPTION**

20320 CX For `asctime()`: The functionality described on this reference page is aligned with the ISO C  
 20321 standard. Any conflict between the requirements described here and the ISO C standard is  
 20322 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

20323 The `asctime()` function shall convert the broken-down time in the structure pointed to by `timeptr`  
 20324 into a string in the form:

```
20325 Sun Sep 16 01:03:52 1973\n\0
```

20326 using the equivalent of the following algorithm:

```
20327 char *asctime(const struct tm *timeptr)
20328 {
20329     static char wday_name[7][3] = {
20330         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
20331     };
20332     static char mon_name[12][3] = {
20333         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
20334         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
20335     };
20336     static char result[26];
20337     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
20338         wday_name[timeptr->tm_wday],
20339         mon_name[timeptr->tm_mon],
20340         timeptr->tm_mday, timeptr->tm_hour,
20341         timeptr->tm_min, timeptr->tm_sec,
20342         1900 + timeptr->tm_year);
20343     return result;
20344 }
```

20345 However, the behavior is undefined if `timeptr->tm_wday` or `timeptr->tm_mon` are not within the  
 20346 normal ranges as defined in `<time.h>`, or if `timeptr->tm_year` exceeds `{INT_MAX}-1990`, or if the  
 20347 above algorithm would attempt to generate more than 26 bytes of output (including the  
 20348 terminating null).

20349 The `tm` structure is defined in the `<time.h>` header.

20350 CX The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static  
 20351 objects: a broken-down time structure and an array of type `char`. Execution of any of the  
 20352 functions may overwrite the information returned in either of these objects by any of the other  
 20353 functions.

20354 The `asctime()` function need not be thread-safe.

20355 The `asctime_r()` function shall convert the broken-down time in the structure pointed to by `tm`  
 20356 into a string (of the same form as that returned by `asctime()`), and with the same undefined  
 20357 behavior when input or output is out of range) that is placed in the user-supplied buffer pointed

20358 to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

#### 20359 RETURN VALUE

20360 CX Upon successful completion, *asctime()* shall return a pointer to the string. If the function is  
20361 unsuccessful, it shall return NULL.

20362 Upon successful completion, *asctime\_r()* shall return a pointer to a character string containing  
20363 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,  
20364 it shall return NULL.

#### 20365 ERRORS

20366 No errors are defined.

#### 20367 EXAMPLES

20368 None.

#### 20369 APPLICATION USAGE

20370 These functions are included only for compatibility with older implementations. They have  
20371 undefined behavior if the resulting string would be too long, so the use of these functions  
20372 should be discouraged. On implementations that do not detect output string length overflow, it  
20373 is possible to overflow the output buffers in such a way as to cause applications to fail, or  
20374 possible system security violations. Also, these functions do not support localized date and time  
20375 formats. To avoid these problems, applications should use *strptime()* to generate strings from  
20376 broken-down times.

20377 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

20378 The *asctime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
20379 of possibly using a static data area that may be overwritten by each call.

#### 20380 RATIONALE

20381 The standard developers decided to mark the *asctime()* and *asctime\_r()* functions obsolescent  
20382 even though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C  
20383 standard also provides the *strptime()* function which can be used to avoid these problems.

#### 20384 FUTURE DIRECTIONS

20385 These functions may be removed in a future version.

#### 20386 SEE ALSO

20387 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime\_r()*, *time()*, *utime()*

20388 XBD <time.h>

#### 20389 CHANGE HISTORY

20390 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 20391 Issue 5

20392 Normative text previously in the APPLICATION USAGE section is moved to the  
20393 DESCRIPTION.

20394 The *asctime\_r()* function is included for alignment with the POSIX Threads Extension.

20395 A note indicating that the *asctime()* function need not be reentrant is added to the  
20396 DESCRIPTION.

#### 20397 Issue 6

20398 The *asctime\_r()* function is marked as part of the Thread-Safe Functions option.

20399 Extensions beyond the ISO C standard are marked.

20400 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

**asctime()**

- 20401 its avoidance of possibly using a static data area.
- 20402 The DESCRIPTION of *asctime\_r()* is updated to describe the format of the string returned.
- 20403 The **restrict** keyword is added to the *asctime\_r()* prototype for alignment with the  
20404 ISO/IEC 9899: 1999 standard
- 20405 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in  
20406 the RETURN VALUE section requiring that if the *asctime()* function is unsuccessful it returns  
20407 NULL.
- 20408 **Issue 7**
- 20409 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.
- 20410 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 20411 The *asctime\_r()* function is moved from the Thread-Safe Functions option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

20412 **NAME**

20413 asin, asinf, asinl — arc sine function

20414 **SYNOPSIS**

```
20415 #include <math.h>
20416 double asin(double x);
20417 float asinf(float x);
20418 long double asinl(long double x);
```

20419 **DESCRIPTION**

20420 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 20421 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20422 volume of POSIX.1-2008 defers to the ISO C standard.

20423 These functions shall compute the principal value of the arc sine of their argument  $x$ . The value  
 20424 of  $x$  should be in the range  $[-1,1]$ .

20425 An application wishing to check for error situations should set *errno* to zero and call  
 20426 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 20427 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 20428 zero, an error has occurred.

20429 **RETURN VALUE**

20430 Upon successful completion, these functions shall return the arc sine of  $x$ , in the range  
 20431  $[-\pi/2, \pi/2]$  radians.

20432 **MX** For finite values of  $x$  not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 20433 supported), or an implementation-defined value shall be returned.

20434 **MX** If  $x$  is NaN, a NaN shall be returned.

20435 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

20436 If  $x$  is  $\pm \text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 20437 defined value shall be returned.

20438 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

20439 **ERRORS**

20440 These functions shall fail if:

20441 **MX** Domain Error The  $x$  argument is finite and is not in the range  $[-1,1]$ , or is  $\pm \text{Inf}$ .

20442 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20443 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 20444 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 20445 shall be raised.

20446 These functions may fail if:

20447 **MX** Range Error The value of  $x$  is subnormal.

20448 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20449 then *errno* shall be set to [ERANGE]. If the integer expression  
 20450 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 20451 floating-point exception shall be raised.

**asin()**20452 **EXAMPLES**

20453 None.

20454 **APPLICATION USAGE**20455 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
20456 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.20457 **RATIONALE**

20458 None.

20459 **FUTURE DIRECTIONS**

20460 None.

20461 **SEE ALSO**20462 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [sin\(\)](#)20463 XBD Section 4.19 (on page 116), [<math.h>](#)20464 **CHANGE HISTORY**

20465 First released in Issue 1. Derived from Issue 1 of the SVID.

20466 **Issue 5**20467 The DESCRIPTION is updated to indicate how an application should check for an error. This  
20468 text was previously published in the APPLICATION USAGE section.20469 **Issue 6**20470 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.20471 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
20472 revised to align with the ISO/IEC 9899:1999 standard.20473 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
20474 marked.

20475 **NAME**

20476 asinh, asinhf, asinhl — inverse hyperbolic sine functions

20477 **SYNOPSIS**

```
20478 #include <math.h>
20479 double asinh(double x);
20480 float asinhf(float x);
20481 long double asinhl(long double x);
```

20482 **DESCRIPTION**

20483 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 20484 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20485 volume of POSIX.1-2008 defers to the ISO C standard.

20486 These functions shall compute the inverse hyperbolic sine of their argument *x*.

20487 An application wishing to check for error situations should set *errno* to zero and call  
 20488 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 20489 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 20490 zero, an error has occurred.

20491 **RETURN VALUE**

20492 Upon successful completion, these functions shall return the inverse hyperbolic sine of their  
 20493 argument.

20494 **MX** If *x* is NaN, a NaN shall be returned.

20495 If *x* is  $\pm 0$ , or  $\pm \text{Inf}$ , *x* shall be returned.

20496 If *x* is subnormal, a range error may occur and *x* should be returned.

20497 **ERRORS**

20498 These functions may fail if:

20499 **MX** **Range Error** The value of *x* is subnormal.

20500 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20501 then *errno* shall be set to [ERANGE]. If the integer expression  
 20502 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 20503 floating-point exception shall be raised.

20504 **EXAMPLES**

20505 None.

20506 **APPLICATION USAGE**

20507 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 20508 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20509 **RATIONALE**

20510 None.

20511 **FUTURE DIRECTIONS**

20512 None.

20513 **SEE ALSO**

20514 *feclearexcept()*, *fetestexcept()*, *sinh()*

20515 XBD Section 4.19 (on page 116), [<math.h>](#)

**asinh()**20516 **CHANGE HISTORY**

20517 First released in Issue 4, Version 2.

20518 **Issue 5**

20519 Moved from X/OPEN UNIX extension to BASE.

20520 **Issue 6**20521 The *asinh()* function is no longer marked as an extension.20522 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.  
2052320524 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
20525 revised to align with the ISO/IEC 9899:1999 standard.20526 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
20527 marked.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***asinl()**20528 **NAME**20529        **asinl** — arc sine function20530 **SYNOPSIS**

20531        #include &lt;math.h&gt;

20532        long double asinl(long double *x*);20533 **DESCRIPTION**20534        Refer to *asin()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**assert()**20535 **NAME**20536 `assert` — insert program diagnostics20537 **SYNOPSIS**20538 `#include <assert.h>`20539 `void assert(scalar expression);`20540 **DESCRIPTION**

20541 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 20542 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20543 volume of POSIX.1-2008 defers to the ISO C standard.

20544 The `assert()` macro shall insert diagnostics into programs; it shall expand to a **void** expression.  
 20545 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal  
 20546 to 0), `assert()` shall write information about the particular call that failed on `stderr` and shall call  
 20547 `abort()`.

20548 The information written about the call that failed shall include the text of the argument, the  
 20549 name of the source file, the source file line number, and the name of the enclosing function; the  
 20550 latter are, respectively, the values of the preprocessing macros `_FILE_` and `__LINE_` and of  
 20551 the identifier `__func_`.

20552 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the  
 20553 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,  
 20554 shall stop assertions from being compiled into the program.

20555 **RETURN VALUE**20556 The `assert()` macro shall not return a value.20557 **ERRORS**

20558 No errors are defined.

20559 **EXAMPLES**

20560 None.

20561 **APPLICATION USAGE**

20562 None.

20563 **RATIONALE**

20564 None.

20565 **FUTURE DIRECTIONS**

20566 None.

20567 **SEE ALSO**20568 `abort()`, `stdin`20569 `XBD <assert.h>`20570 **CHANGE HISTORY**

20571 First released in Issue 1. Derived from Issue 1 of the SVID.

20572 **Issue 6**20573 The prototype for the *expression* argument to `assert()` is changed from **int** to **scalar** for alignment  
 20574 with the ISO/IEC 9899:1999 standard.20575 The DESCRIPTION of `assert()` is updated for alignment with the ISO/IEC 9899:1999 standard.

20576 **NAME**

20577 atan, atanf, atanl — arc tangent function

20578 **SYNOPSIS**

```
20579 #include <math.h>
20580 double atan(double x);
20581 float atanf(float x);
20582 long double atanl(long double x);
```

20583 **DESCRIPTION**

20584 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20585 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20586 volume of POSIX.1-2008 defers to the ISO C standard.

20587 These functions shall compute the principal value of the arc tangent of their argument  $x$ .

20588 An application wishing to check for error situations should set *errno* to zero and call  
 20589 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 20590 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 20591 zero, an error has occurred.

20592 **RETURN VALUE**

20593 Upon successful completion, these functions shall return the arc tangent of  $x$  in the range  
 20594  $[-\pi/2, \pi/2]$  radians.

20595 MX If  $x$  is NaN, a NaN shall be returned.

20596 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

20597 If  $x$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.

20598 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

20599 **ERRORS**

20600 These functions may fail if:

20601 MX Range Error The value of  $x$  is subnormal.

20602 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20603 then *errno* shall be set to [ERANGE]. If the integer expression  
 20604 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 20605 floating-point exception shall be raised.

20606 **EXAMPLES**

20607 None.

20608 **APPLICATION USAGE**

20609 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 20610 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20611 **RATIONALE**

20612 None.

20613 **FUTURE DIRECTIONS**

20614 None.

20615 **SEE ALSO**

20616 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

20617 XBD Section 4.19 (on page 116), [<math.h>](#)

**atan()**20618 **CHANGE HISTORY**

20619 First released in Issue 1. Derived from Issue 1 of the SVID.

20620 **Issue 5**

20621 The DESCRIPTION is updated to indicate how an application should check for an error. This  
20622 text was previously published in the APPLICATION USAGE section.

20623 **Issue 6**

20624 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

20625 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
20626 revised to align with the ISO/IEC 9899:1999 standard.

20627 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
20628 marked.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

20629 **NAME**

20630 atan2, atan2f, atan2l — arc tangent functions

20631 **SYNOPSIS**

20632 #include &lt;math.h&gt;

20633 double atan2(double y, double x);

20634 float atan2f(float y, float x);

20635 long double atan2l(long double y, long double x);

20636 **DESCRIPTION**20637 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
20638 conflict between the requirements described here and the ISO C standard is unintentional. This  
20639 volume of POSIX.1-2008 defers to the ISO C standard.20640 These functions shall compute the principal value of the arc tangent of  $y/x$ , using the signs of  
20641 both arguments to determine the quadrant of the return value.20642 An application wishing to check for error situations should set *errno* to zero and call  
20643 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
20644 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
20645 zero, an error has occurred.20646 **RETURN VALUE**20647 Upon successful completion, these functions shall return the arc tangent of  $y/x$  in the range  
20648  $[-\pi, \pi]$  radians.20649 If  $y$  is  $\pm 0$  and  $x$  is  $< 0$ ,  $\pm\pi$  shall be returned.20650 If  $y$  is  $\pm 0$  and  $x$  is  $> 0$ ,  $\pm 0$  shall be returned.20651 If  $y$  is  $< 0$  and  $x$  is  $\pm 0$ ,  $-\pi/2$  shall be returned.20652 If  $y$  is  $> 0$  and  $x$  is  $\pm 0$ ,  $\pi/2$  shall be returned.20653 If  $x$  is 0, a pole error shall not occur.20654 **MX** If either  $x$  or  $y$  is NaN, a NaN shall be returned.20655 If the result underflows, a range error may occur and  $y/x$  should be returned.20656 If  $y$  is  $\pm 0$  and  $x$  is  $-0$ ,  $\pm\pi$  shall be returned.20657 If  $y$  is  $\pm 0$  and  $x$  is  $+0$ ,  $\pm 0$  shall be returned.20658 For finite values of  $\pm y > 0$ , if  $x$  is  $-\text{Inf}$ ,  $\pm\pi$  shall be returned.20659 For finite values of  $\pm y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $\pm 0$  shall be returned.20660 For finite values of  $x$ , if  $y$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.20661 If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $-\text{Inf}$ ,  $\pm 3\pi/4$  shall be returned.20662 If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $+\text{Inf}$ ,  $\pm\pi/4$  shall be returned.

20663 If both arguments are 0, a domain error shall not occur.

20664 **ERRORS**

20665 These functions may fail if:

20666 **MX** **Range Error** The result underflows.20667 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
20668 then *errno* shall be set to [ERANGE]. If the integer expression

**atan2()**

20669 *(math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 20670 floating-point exception shall be raised.

20671 **EXAMPLES**20672 **Converting Cartesian to Polar Coordinates System**

20673 The function below uses *atan2()* to convert a 2d vector expressed in cartesian coordinates  $(x,y)$  to  
 20674 the polar coordinates  $(rho,theta)$ . There are other ways to compute the angle *theta*, using *asin()*  
 20675 *acos()*, or *atan()*. However, *atan2()* presents here two advantages:

- 20676 • The angle's quadrant is automatically determined.
- 20677 • The singular cases  $(0,y)$  are taken into account.

20678 Finally, this example uses *hypot()* rather than *sqrt()* since it is better for special cases; see *hypot()*  
 20679 for more information.

```
20680 #include <math.h>
20681 void
20682 cartesian_to_polar(const double x, const double y,
20683                  double *rho, double *theta)
20684 {
20685     {
20686         *rho = hypot (x,y); /* better than sqrt(x*x+y*y) */
20687         *theta = atan2 (y,x);
20688     }

```

20689 **APPLICATION USAGE**

20690 On error, the expressions *(math\_errhandling* & MATH\_ERRNO) and *(math\_errhandling* &  
 20691 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20692 **RATIONALE**

20693 None.

20694 **FUTURE DIRECTIONS**

20695 None.

20696 **SEE ALSO**

20697 *acos()*, *asin()*, *atan()*, *feclearexcept()*, *fetestexcept()*, *hypot()*, *isnan()*, *sqrt()*, *tan()*

20698 XBD Section 4.19 (on page 116), **<math.h>**

20699 **CHANGE HISTORY**

20700 First released in Issue 1. Derived from Issue 1 of the SVID.

20701 **Issue 5**

20702 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 20703 text was previously published in the APPLICATION USAGE section.

20704 **Issue 6**

20705 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999  
 20706 standard.

20707 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 20708 revised to align with the ISO/IEC 9899:1999 standard, and the IEC 60559:1989 standard  
 20709 floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

20710  
20711

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**atanf()**20712 **NAME**

20713            atanf — arc tangent function

20714 **SYNOPSIS**

20715            #include &lt;math.h&gt;

20716            float atanf(float x);

20717 **DESCRIPTION**20718            Refer to *atan()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

20719 **NAME**

20720 atanh, atanhf, atanhf — inverse hyperbolic tangent functions

20721 **SYNOPSIS**

```
20722 #include <math.h>
20723 double atanh(double x);
20724 float atanhf(float x);
20725 long double atanhf(long double x);
```

20726 **DESCRIPTION**

20727 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20728 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20729 volume of POSIX.1-2008 defers to the ISO C standard.

20730 These functions shall compute the inverse hyperbolic tangent of their argument  $x$ .

20731 An application wishing to check for error situations should set *errno* to zero and call  
 20732 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 20733 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 20734 zero, an error has occurred.

20735 **RETURN VALUE**

20736 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their  
 20737 argument.

20738 If  $x$  is  $\pm 1$ , a pole error shall occur, and *atanh*( $x$ ), *atanhf*( $x$ ), and *atanhl*( $x$ ) shall return the value of the  
 20739 macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively, with the same sign as the  
 20740 correct value of the function.

20741 MX For finite  $|x| > 1$ , a domain error shall occur, and either a NaN (if supported), or an  
 20742 implementation-defined value shall be returned.

20743 MX If  $x$  is NaN, a NaN shall be returned.

20744 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

20745 If  $x$  is  $\pm \text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 20746 defined value shall be returned.

20747 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

20748 **ERRORS**

20749 These functions shall fail if:

20750 MX **Domain Error** The  $x$  argument is finite and not in the range  $[-1,1]$ , or is  $\pm \text{Inf}$ .

20751 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20752 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 20753 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 20754 shall be raised.

20755 **Pole Error** The  $x$  argument is  $\pm 1$ .

20756 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20757 then *errno* shall be set to [ERANGE]. If the integer expression  
 20758 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 20759 floating-point exception shall be raised.

**atanh()**

20760 These functions may fail if:

20761 MX **Range Error** The value of  $x$  is subnormal.

20762 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 20763 then *errno* shall be set to [ERANGE]. If the integer expression  
 20764 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 20765 floating-point exception shall be raised.

20766 **EXAMPLES**

20767 None.

20768 **APPLICATION USAGE**

20769 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 20770 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20771 **RATIONALE**

20772 None.

20773 **FUTURE DIRECTIONS**

20774 None.

20775 **SEE ALSO**

20776 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [tanh\(\)](#)

20777 XBD [Section 4.19](#) (on page 116), [<math.h>](#)

20778 **CHANGE HISTORY**

20779 First released in Issue 4, Version 2.

20780 **Issue 5**

20781 Moved from X/OPEN UNIX extension to BASE.

20782 **Issue 6**

20783 The *atanh()* function is no longer marked as an extension.

20784 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999  
 20785 standard.

20786 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 20787 revised to align with the ISO/IEC 9899:1999 standard.

20788 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 20789 marked.

*System Interfaces***atanl()**20790 **NAME**

20791       atanl — arc tangent function

20792 **SYNOPSIS**

20793       #include &lt;math.h&gt;

20794       long double atanl(long double x);

20795 **DESCRIPTION**20796       Refer to *atan()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**atexit()**20797 **NAME**20798 `atexit` — register a function to run at process termination20799 **SYNOPSIS**

```
20800 #include <stdlib.h>
20801 int atexit(void (*func)(void));
```

20802 **DESCRIPTION**

20803 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20804 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20805 volume of POSIX.1-2008 defers to the ISO C standard.

20806 The `atexit()` function shall register the function pointed to by `func`, to be called without  
 20807 arguments at normal program termination. At normal program termination, all functions  
 20808 registered by the `atexit()` function shall be called, in the reverse order of their registration, except  
 20809 that a function is called after any previously registered functions that had already been called at  
 20810 the time it was registered. Normal termination occurs either by a call to `exit()` or a return from  
 20811 `main()`.

20812 At least 32 functions can be registered with `atexit()`.

20813 CX After a successful call to any of the `exec` functions, any functions previously registered by `atexit()`  
 20814 shall no longer be registered.

20815 **RETURN VALUE**

20816 Upon successful completion, `atexit()` shall return 0; otherwise, it shall return a non-zero value.

20817 **ERRORS**

20818 No errors are defined.

20819 **EXAMPLES**

20820 None.

20821 **APPLICATION USAGE**

20822 The functions registered by a call to `atexit()` must return to ensure that all registered functions  
 20823 are called.

20824 The application should call `sysconf()` to obtain the value of `{ATEXIT_MAX}`, the number of  
 20825 functions that can be registered. There is no way for an application to tell how many functions  
 20826 have already been registered with `atexit()`.

20827 Since the behavior is undefined if the `exit()` function is called more than once, portable  
 20828 applications calling `atexit()` must ensure that the `exit()` function is not called at normal process  
 20829 termination when all functions registered by the `atexit()` function are called.

20830 All functions registered by the `atexit()` function are called at normal process termination, which  
 20831 occurs by a call to the `exit()` function or a return from `main()` or on the last thread termination,  
 20832 when the behavior is as if the implementation called `exit()` with a zero argument at thread  
 20833 termination time.

20834 If, at normal process termination, a function registered by the `atexit()` function is called and a  
 20835 portable application needs to stop further `exit()` processing, it must call the `_exit()` function or  
 20836 the `_Exit()` function or one of the functions which cause abnormal process termination.

20837 **RATIONALE**

20838 None.

20839 **FUTURE DIRECTIONS**

20840 None.

20841 **SEE ALSO**20842 *exec*, *exit()*, *sysconf()*20843 XBD `<stdlib.h>`20844 **CHANGE HISTORY**

20845 First released in Issue 4. Derived from the ANSI C standard.

20846 **Issue 6**

20847 Extensions beyond the ISO C standard are marked.

20848 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

20849 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification  
20850 to the APPLICATION USAGE section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**atof()**20851 **NAME**20852            **atof** — convert a string to a double-precision number20853 **SYNOPSIS**

20854            #include &lt;stdlib.h&gt;

20855            double atof(const char \*str);

20856 **DESCRIPTION**

20857 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 20858 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20859 volume of POSIX.1-2008 defers to the ISO C standard.

20860            The call *atof(str)* shall be equivalent to:

20861            strtod(str, (char \*\*)NULL),

20862 except that the handling of errors may differ. If the value cannot be represented, the behavior is  
 20863 undefined.

20864 **RETURN VALUE**20865            The *atof()* function shall return the converted value if the value can be represented.20866 **ERRORS**

20867            No errors are defined.

20868 **EXAMPLES**

20869            None.

20870 **APPLICATION USAGE**

20871            The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in  
 20872 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()*  
 20873 is not required to perform any error checking.

20874 **RATIONALE**

20875            None.

20876 **FUTURE DIRECTIONS**

20877            None.

20878 **SEE ALSO**20879            *strtod()*

20880            XBD &lt;stdlib.h&gt;

20881 **CHANGE HISTORY**

20882            First released in Issue 1. Derived from Issue 1 of the SVID.

20883 **NAME**

20884        atoi — convert a string to an integer

20885 **SYNOPSIS**

```
20886        #include <stdlib.h>
20887        int atoi(const char *str);
```

20888 **DESCRIPTION**

20889 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 20890 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20891 volume of POSIX.1-2008 defers to the ISO C standard.

20892        The call *atoi(str)* shall be equivalent to:

```
20893        (int) strtol(str, (char **)NULL, 10)
```

20894        except that the handling of errors may differ. If the value cannot be represented, the behavior is  
 20895 undefined.

20896 **RETURN VALUE**

20897        The *atoi()* function shall return the converted value if the value can be represented.

20898 **ERRORS**

20899        No errors are defined.

20900 **EXAMPLES**20901        **Converting an Argument**

20902        The following example checks for proper usage of the program. If there is an argument and the  
 20903 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program  
 20904 has a valid number of minutes to wait for an event.

```
20905        #include <stdlib.h>
20906        #include <stdio.h>
20907        ...
20908        int minutes_to_event;
20909        ...
20910        if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
20911            fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
20912        }
20913        ...
```

20914 **APPLICATION USAGE**

20915        The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in  
 20916 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is  
 20917 not required to perform any error checking.

20918 **RATIONALE**

20919        None.

20920 **FUTURE DIRECTIONS**

20921        None.

20922 **SEE ALSO**

20923        *strtol()*

20924        XBD <stdlib.h>

**atoi()**

20925 **CHANGE HISTORY**

20926 First released in Issue 1. Derived from Issue 1 of the SVID.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

20927 **NAME**

20928 atol, atoll — convert a string to a long integer

20929 **SYNOPSIS**

20930 #include &lt;stdlib.h&gt;

20931 long atol(const char \*str);

20932 long long atoll(const char \*nptr);

20933 **DESCRIPTION**

20934 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20935 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20936 volume of POSIX.1-2008 defers to the ISO C standard.

20937 The call *atol(str)* shall be equivalent to:20938 `strtol(str, (char **)NULL, 10)`20939 The call *atoll(nptr)* shall be equivalent to:20940 `strtoll(nptr, (char **)NULL, 10)`20941 except that the handling of errors may differ. If the value cannot be represented, the behavior is  
 20942 undefined.20943 **RETURN VALUE**

20944 These functions shall return the converted value if the value can be represented.

20945 **ERRORS**

20946 No errors are defined.

20947 **EXAMPLES**

20948 None.

20949 **APPLICATION USAGE**

20950 The *atol()* function is subsumed by *strtol()* but is retained because it is used extensively in  
 20951 existing code. If the number is not known to be in range, *strtol()* should be used because *atol()* is  
 20952 not required to perform any error checking.

20953 **RATIONALE**

20954 None.

20955 **FUTURE DIRECTIONS**

20956 None.

20957 **SEE ALSO**20958 [strtol\(\)](#)20959 XBD [<stdlib.h>](#)20960 **CHANGE HISTORY**

20961 First released in Issue 1. Derived from Issue 1 of the SVID.

20962 **Issue 6**20963 The *atoll()* function is added for alignment with the ISO/IEC 9899:1999 standard.20964 **Issue 7**20965 SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of *atoll()*.

# basename()

20966 **NAME**

20967 `basename` — return the last component of a pathname

20968 **SYNOPSIS**

```
20969 XSI #include <libgen.h>
20970 char *basename(char *path);
```

20971 **DESCRIPTION**

20972 The `basename()` function shall take the pathname pointed to by `path` and return a pointer to the  
 20973 final component of the pathname, deleting any trailing `'/'` characters.

20974 If the string pointed to by `path` consists entirely of the `'/'` character, `basename()` shall return a  
 20975 pointer to the string `"/"`. If the string pointed to by `path` is exactly `"/"`, it is implementation-  
 20976 defined whether `'/'` or `"/"` is returned.

20977 If `path` is a null pointer or points to an empty string, `basename()` shall return a pointer to the  
 20978 string  `"."`.

20979 The `basename()` function may modify the string pointed to by `path`, and may return a pointer to  
 20980 static storage that may then be overwritten by a subsequent call to `basename()`.

20981 The `basename()` function need not be thread-safe.

20982 **RETURN VALUE**

20983 The `basename()` function shall return a pointer to the final component of `path`.

20984 **ERRORS**

20985 No errors are defined.

20986 **EXAMPLES**

20987 **Using `basename()`**

20988 The following program fragment returns a pointer to the value `lib`, which is the base name of  
 20989 `/usr/lib`.

```
20990 #include <libgen.h>
20991 ...
20992 char *name = "/usr/lib";
20993 char *base;
20994 base = basename(name);
20995 ...
```

20996 **Sample Input and Output Strings for `basename()`**

20997 In the following table, the input string is the value pointed to by `path`, and the output string is  
 20998 the return value of the `basename()` function.

| Input String               | Output String      |
|----------------------------|--------------------|
| <code>"/usr/lib"</code>    | <code>"lib"</code> |
| <code>"/usr/"</code>       | <code>"usr"</code> |
| <code>"/"</code>           | <code>"/"</code>   |
| <code>"/" / "</code>       | <code>"/"</code>   |
| <code>"/usr//lib//"</code> | <code>"lib"</code> |

21005 **APPLICATION USAGE**

21006 None.

21007 **RATIONALE**

21008 None.

21009 **FUTURE DIRECTIONS**

21010 None.

21011 **SEE ALSO**21012 *dirname()*

21013 XBD &lt;libgen.h&gt;

21014 XCU *basename*21015 **CHANGE HISTORY**

21016 First released in Issue 4, Version 2.

21017 **Issue 5**

21018 Moved from X/OPEN UNIX extension to BASE.

21019 Normative text previously in the APPLICATION USAGE section is moved to the  
21020 DESCRIPTION.

21021 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

21022 **Issue 6**

21023 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

21024 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the  
21025 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.21026 **Issue 7**

21027 Austin Group Interpretation 1003.1-2001 #156 is applied.

**bind()**21028 **NAME**21029 `bind` — bind a name to a socket21030 **SYNOPSIS**21031 `#include <sys/socket.h>`21032 `int bind(int socket, const struct sockaddr *address,`  
21033 `socklen_t address_len);`21034 **DESCRIPTION**21035 The `bind()` function shall assign a local socket address `address` to a socket identified by descriptor  
21036 `socket` that has no local socket address assigned. Sockets created with the `socket()` function are  
21037 initially unnamed; they are identified only by their address family.21038 The `bind()` function takes the following arguments:21039 `socket` Specifies the file descriptor of the socket to be bound.21040 `address` Points to a **sockaddr** structure containing the address to be bound to the  
21041 socket. The length and format of the address depend on the address family of  
21042 the socket.21043 `address_len` Specifies the length of the **sockaddr** structure pointed to by the `address`  
21044 argument.21045 The socket specified by `socket` may require the process to have appropriate privileges to use the  
21046 `bind()` function.21047 If the socket address cannot be assigned immediately and `O_NONBLOCK` is set for the file  
21048 descriptor for the socket, `bind()` shall fail and set `errno` to `[EINPROGRESS]`, but the assignment  
21049 request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent  
21050 calls to `bind()` for the same socket, before the assignment is completed, shall fail and set `errno` to  
21051 `[EALREADY]`.21052 When the assignment has been performed asynchronously, `pselect()`, `select()`, and `poll()` shall  
21053 indicate that the file descriptor for the socket is ready for reading and writing.21054 **RETURN VALUE**21055 Upon successful completion, `bind()` shall return 0; otherwise, `-1` shall be returned and `errno` set  
21056 to indicate the error.21057 **ERRORS**21058 The `bind()` function shall fail if:21059 `[EADDRINUSE]` The specified address is already in use.21060 `[EADDRNOTAVAIL]`

21061 The specified address is not available from the local machine.

21062 `[EAFNOSUPPORT]`21063 The specified address is not a valid address for the address family of the  
21064 specified socket.21065 `[EALREADY]` An assignment request is already in progress for the specified socket.21066 `[EBADF]` The `socket` argument is not a valid file descriptor.21067 `[EINPROGRESS]` `O_NONBLOCK` is set for the file descriptor for the socket and the assignment  
21068 cannot be immediately performed; the assignment shall be performed  
21069 asynchronously.

|       |                            |                                                                                                                                                                                                                                                                                                                                                                            |
|-------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21070 | [EINVAL]                   | The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.                                                                                                                                                                                                                                   |
| 21071 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21072 | [ENOBUFS]                  | Insufficient resources were available to complete the call.                                                                                                                                                                                                                                                                                                                |
| 21073 | [ENOTSOCK]                 | The <i>socket</i> argument does not refer to a socket.                                                                                                                                                                                                                                                                                                                     |
| 21074 | [EOPNOTSUPP]               | The socket type of the specified socket does not support binding to an address.                                                                                                                                                                                                                                                                                            |
| 21075 |                            | If the address family of the socket is AF_UNIX, then <i>bind()</i> shall fail if:                                                                                                                                                                                                                                                                                          |
| 21076 | [EACCES]                   | A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.                                                                                                                                                                                                                   |
| 21077 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21078 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21079 | [EDESTADDRREQ] or [EISDIR] |                                                                                                                                                                                                                                                                                                                                                                            |
| 21080 |                            | The <i>address</i> argument is a null pointer.                                                                                                                                                                                                                                                                                                                             |
| 21081 | [EIO]                      | An I/O error occurred.                                                                                                                                                                                                                                                                                                                                                     |
| 21082 | [ELOOP]                    | A loop exists in symbolic links encountered during resolution of the pathname in <i>address</i> .                                                                                                                                                                                                                                                                          |
| 21083 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21084 | [ENAMETOOLONG]             |                                                                                                                                                                                                                                                                                                                                                                            |
| 21085 |                            | The length of a component of a pathname is longer than {NAME_MAX}.                                                                                                                                                                                                                                                                                                         |
| 21086 | [ENOENT]                   | A component of the pathname does not name an existing file or the pathname is an empty string.                                                                                                                                                                                                                                                                             |
| 21087 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21088 | [ENOTDIR]                  | A component of the path prefix of the pathname in <i>address</i> is not a directory, or the pathname in <i>address</i> contains at least one non- <i>&lt;slash&gt;</i> character and ends with one or more trailing <i>&lt;slash&gt;</i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory. |
| 21089 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21090 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21091 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21092 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21093 | [EROFS]                    | The name would reside on a read-only file system.                                                                                                                                                                                                                                                                                                                          |
| 21094 |                            | The <i>bind()</i> function may fail if:                                                                                                                                                                                                                                                                                                                                    |
| 21095 | [EACCES]                   | The specified address is protected and the current user does not have permission to bind to it.                                                                                                                                                                                                                                                                            |
| 21096 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21097 | [EINVAL]                   | The <i>address_len</i> argument is not a valid length for the address family.                                                                                                                                                                                                                                                                                              |
| 21098 | [EISCONN]                  | The socket is already connected.                                                                                                                                                                                                                                                                                                                                           |
| 21099 | [ELOOP]                    | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .                                                                                                                                                                                                                                                              |
| 21100 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21101 | [ENAMETOOLONG]             |                                                                                                                                                                                                                                                                                                                                                                            |
| 21102 |                            | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.                                                                                                                                                                                                              |
| 21103 |                            |                                                                                                                                                                                                                                                                                                                                                                            |
| 21104 |                            |                                                                                                                                                                                                                                                                                                                                                                            |

**bind()**21105 **EXAMPLES**

21106 The following code segment shows how to create a socket and bind it to a name in the AF\_UNIX  
21107 domain.

```
21108 #define MY_SOCKET_PATH "/somepath"
21109
21109 int sfd;
21110 struct sockaddr_un my_addr;
21111
21111 sfd = socket(AF_UNIX, SOCK_STREAM, 0);
21112 if (sfd == -1)
21113     /* Handle error */;
21114
21114 memset(&my_addr, '\0', sizeof(struct sockaddr_un));
21115     /* Clear structure */
21116 my_addr.sun_family = AF_UNIX;
21117 strncpy(my_addr.sun_path, MY_SOCKET_PATH, sizeof(my_addr.sun_path) - 1);
21118
21118 if (bind(sfd, (struct sockaddr *) &my_addr,
21119         sizeof(struct sockaddr_un)) == -1)
21120     /* Handle error */;
```

21121 **APPLICATION USAGE**

21122 An application program can retrieve the assigned socket name with the *getsockname()* function.

21123 **RATIONALE**

21124 None.

21125 **FUTURE DIRECTIONS**

21126 None.

21127 **SEE ALSO**

21128 *connect()*, *getsockname()*, *listen()*, *socket()*

21129 XBD <sys/socket.h>

21130 **CHANGE HISTORY**

21131 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

21132 **Issue 7**

21133 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS]  
21134 error to become a “shall fail” error.

21135 Austin Group Interpretation 1003.1-2001 #143 is applied.

21136 SD5-XSH-ERN-185 is applied.

21137 An example is added.



**bsearch()**

```

21182     char *string;
21183     int length;
21184 };
21185 struct node table[TABSIZE];    /* Table to be searched. */
21186     .
21187     .
21188     .
21189 {
21190     struct node *node_ptr, node;
21191     /* Routine to compare 2 nodes. */
21192     int node_compare(const void *, const void *);
21193     .
21194     .
21195     .
21196     while (scanf("%ms", &node.string) != EOF) {
21197         node_ptr = (struct node *)bsearch((void *)(&node),
21198             (void *)table, TABSIZE,
21199             sizeof(struct node), node_compare);
21200         if (node_ptr != NULL) {
21201             (void)printf("string = %20s, length = %d\n",
21202                 node_ptr->string, node_ptr->length);
21203         } else {
21204             (void)printf("not found: %s\n", node.string);
21205         }
21206         free(node.string);
21207     }
21208 }
21209 /*
21210     This routine compares two nodes based on an
21211     alphabetical ordering of the string field.
21212 */
21213 int
21214 node_compare(const void *node1, const void *node2)
21215 {
21216     return strcmp(((const struct node *)node1)->string,
21217         ((const struct node *)node2)->string);
21218 }

```

**APPLICATION USAGE**

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

**RATIONALE**

The requirement that the second argument (hereafter referred to as *p*) to the comparison function is a pointer to an element of the array implies that for every call all of the following expressions are non-zero:

```

((char *)p - (char *(base) % width) == 0
(char *)p >= (char *)base

```

21231 (char \*)p < (char \*)base + nel \* width

21232 **FUTURE DIRECTIONS**

21233 None.

21234 **SEE ALSO**

21235 *hcreate()*, *lsearch()*, *qsort()*, *tdelete()*

21236 XBD <stdlib.h>

21237 **CHANGE HISTORY**

21238 First released in Issue 1. Derived from Issue 1 of the SVID.

21239 **Issue 6**

21240 The normative text is updated to avoid use of the term “must” for application requirements.

21241 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the  
21242 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three  
21243 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the  
21244 ISO C standard.

21245 **Issue 7**

21246 The EXAMPLES section is revised.

**btowc()**21247 **NAME**21248 `btowc` — single byte to wide character conversion21249 **SYNOPSIS**

```
21250 #include <stdio.h>
21251 #include <wchar.h>
21252 wint_t btowc(int c);
```

21253 **DESCRIPTION**

21254 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21255 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21256 volume of POSIX.1-2008 defers to the ISO C standard.

21257 The `btowc()` function shall determine whether `c` constitutes a valid (one-byte) character in the  
 21258 initial shift state.

21259 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

21260 **RETURN VALUE**

21261 The `btowc()` function shall return `WEOF` if `c` has the value `EOF` or if (**unsigned char**) `c` does not  
 21262 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the  
 21263 wide-character representation of that character.

21264 **ERRORS**

21265 No errors are defined.

21266 **EXAMPLES**

21267 None.

21268 **APPLICATION USAGE**

21269 None.

21270 **RATIONALE**

21271 None.

21272 **FUTURE DIRECTIONS**

21273 None.

21274 **SEE ALSO**

21275 [wctob\(\)](#)

21276 XBD [<stdio.h>](#) [<wchar.h>](#)

21277 **CHANGE HISTORY**

21278 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 21279 (E).

21280 **NAME**

21281 cabs, cabsf, cabsl — return a complex absolute value

21282 **SYNOPSIS**

```
21283 #include <complex.h>
21284 double cabs(double complex z);
21285 float cabsf(float complex z);
21286 long double cabsl(long double complex z);
```

21287 **DESCRIPTION**

21288 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21289 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21290 volume of POSIX.1-2008 defers to the ISO C standard.

21291 These functions shall compute the complex absolute value (also called norm, modulus, or  
 21292 magnitude) of *z*.

21293 **RETURN VALUE**

21294 These functions shall return the complex absolute value.

21295 **ERRORS**

21296 No errors are defined.

21297 **EXAMPLES**

21298 None.

21299 **APPLICATION USAGE**

21300 None.

21301 **RATIONALE**

21302 None.

21303 **FUTURE DIRECTIONS**

21304 None.

21305 **SEE ALSO**21306 XBD [<complex.h>](#)21307 **CHANGE HISTORY**

21308 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**cacos()**21309 **NAME**21310 `cacos, cacosf, cacosl` — complex arc cosine functions21311 **SYNOPSIS**21312 `#include <complex.h>`21313 `double complex cacos(double complex z);`21314 `float complex cacosf(float complex z);`21315 `long double complex cacosl(long double complex z);`21316 **DESCRIPTION**21317 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21318 conflict between the requirements described here and the ISO C standard is unintentional. This  
21319 volume of POSIX.1-2008 defers to the ISO C standard.21320 These functions shall compute the complex arc cosine of  $z$ , with branch cuts outside the interval  
21321  $[-1, +1]$  along the real axis.21322 **RETURN VALUE**21323 These functions shall return the complex arc cosine value, in the range of a strip mathematically  
21324 unbounded along the imaginary axis and in the interval  $[0, \pi]$  along the real axis.21325 **ERRORS**

21326 No errors are defined.

21327 **EXAMPLES**

21328 None.

21329 **APPLICATION USAGE**

21330 None.

21331 **RATIONALE**

21332 None.

21333 **FUTURE DIRECTIONS**

21334 None.

21335 **SEE ALSO**21336 `ccos()`21337 XBD `<complex.h>`21338 **CHANGE HISTORY**

21339 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21340 **NAME**

21341 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

21342 **SYNOPSIS**

```
21343 #include <complex.h>
21344 double complex cacosh(double complex z);
21345 float complex cacoshf(float complex z);
21346 long double complex cacoshl(long double complex z);
```

21347 **DESCRIPTION**

21348 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21349 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21350 volume of POSIX.1-2008 defers to the ISO C standard.

21351 These functions shall compute the complex arc hyperbolic cosine of  $z$ , with a branch cut at  
 21352 values less than 1 along the real axis.

21353 **RETURN VALUE**

21354 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip  
 21355 of non-negative values along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary axis.

21356 **ERRORS**

21357 No errors are defined.

21358 **EXAMPLES**

21359 None.

21360 **APPLICATION USAGE**

21361 None.

21362 **RATIONALE**

21363 None.

21364 **FUTURE DIRECTIONS**

21365 None.

21366 **SEE ALSO**21367 [ccosh\(\)](#)21368 XBD [<complex.h>](#)21369 **CHANGE HISTORY**

21370 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**cacosl()***System Interfaces*21371 **NAME**

21372           cacosl — complex arc cosine functions

21373 **SYNOPSIS**

21374           #include &lt;complex.h&gt;

21375           long double complex cacosl(long double complex z);

21376 **DESCRIPTION**21377           Refer to *cacos()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

21378 **NAME**

21379        calloc — a memory allocator

21380 **SYNOPSIS**

21381        #include &lt;stdlib.h&gt;

21382        void \*calloc(size\_t *nelem*, size\_t *elsize*);21383 **DESCRIPTION**21384 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
21385 conflict between the requirements described here and the ISO C standard is unintentional. This  
21386 volume of POSIX.1-2008 defers to the ISO C standard.21387        The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose  
21388 size in bytes is *elsize*. The space shall be initialized to all bits 0.21389        The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The  
21390 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
21391 a pointer to any type of object and then used to access such an object or an array of such objects  
21392 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall  
21393 yield a pointer to an object disjoint from any other object. The pointer returned shall point to the  
21394 start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer  
21395 shall be returned. If the size of the space requested is 0, the behavior is implementation-defined:  
21396 the value returned shall be either a null pointer or a unique pointer.21397 **RETURN VALUE**21398        Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to  
21399 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer  
21400 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null  
21401 CX        pointer and set *errno* to indicate the error.21402 **ERRORS**21403        The *calloc()* function shall fail if:

21404 CX        [ENOMEM]        Insufficient memory is available.

21405 **EXAMPLES**

21406        None.

21407 **APPLICATION USAGE**

21408        There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

21409 **RATIONALE**

21410        None.

21411 **FUTURE DIRECTIONS**

21412        None.

21413 **SEE ALSO**21414        *free()*, *malloc()*, *realloc()*

21415        XBD &lt;stdlib.h&gt;

21416 **CHANGE HISTORY**

21417        First released in Issue 1. Derived from Issue 1 of the SVID.

21418 **Issue 6**

21419        Extensions beyond the ISO C standard are marked.

**calloc()**

- 21420 The following new requirements on POSIX implementations derive from alignment with the  
21421 Single UNIX Specification:
- 21422     • The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient  
21423 memory condition occurs.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

21424 **NAME**

21425 carg, cargf, cargl — complex argument functions

21426 **SYNOPSIS**

```
21427 #include <complex.h>
21428 double carg(double complex z);
21429 float cargf(float complex z);
21430 long double cargl(long double complex z);
```

21431 **DESCRIPTION**

21432 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21433 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21434 volume of POSIX.1-2008 defers to the ISO C standard.

21435 These functions shall compute the argument (also called phase angle) of  $z$ , with a branch cut  
 21436 along the negative real axis.

21437 **RETURN VALUE**21438 These functions shall return the value of the argument in the interval  $[-\pi, +\pi]$ .21439 **ERRORS**

21440 No errors are defined.

21441 **EXAMPLES**

21442 None.

21443 **APPLICATION USAGE**

21444 None.

21445 **RATIONALE**

21446 None.

21447 **FUTURE DIRECTIONS**

21448 None.

21449 **SEE ALSO**21450 [cimag\(\)](#), [conj\(\)](#), [cproj\(\)](#)21451 XBD [<complex.h>](#)21452 **CHANGE HISTORY**

21453 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**casin()**21454 **NAME**21455            **casin, casinf, casinl** — complex arc sine functions21456 **SYNOPSIS**

```
21457            #include <complex.h>

21458            double complex casin(double complex z);
21459            float complex casinf(float complex z);
21460            long double complex casinl(long double complex z);
```

21461 **DESCRIPTION**

21462 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 21463 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21464 volume of POSIX.1-2008 defers to the ISO C standard.

21465            These functions shall compute the complex arc sine of  $z$ , with branch cuts outside the interval  
 21466  $[-1, +1]$  along the real axis.

21467 **RETURN VALUE**

21468            These functions shall return the complex arc sine value, in the range of a strip mathematically  
 21469 unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the real axis.

21470 **ERRORS**

21471            No errors are defined.

21472 **EXAMPLES**

21473            None.

21474 **APPLICATION USAGE**

21475            None.

21476 **RATIONALE**

21477            None.

21478 **FUTURE DIRECTIONS**

21479            None.

21480 **SEE ALSO**21481            [csin\(\)](#)21482            XBD [<complex.h>](#)21483 **CHANGE HISTORY**

21484            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21485 **NAME**

21486 casinh, casinhf, casinhl — complex arc hyperbolic sine functions

21487 **SYNOPSIS**

21488 #include &lt;complex.h&gt;

21489 double complex casinh(double complex z);

21490 float complex casinhf(float complex z);

21491 long double complex casinhl(long double complex z);

21492 **DESCRIPTION**21493 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21494 conflict between the requirements described here and the ISO C standard is unintentional. This  
21495 volume of POSIX.1-2008 defers to the ISO C standard.21496 These functions shall compute the complex arc hyperbolic sine of  $z$ , with branch cuts outside the  
21497 interval  $[-i, +i]$  along the imaginary axis.21498 **RETURN VALUE**21499 These functions shall return the complex arc hyperbolic sine value, in the range of a strip  
21500 mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
21501 imaginary axis.21502 **ERRORS**

21503 No errors are defined.

21504 **EXAMPLES**

21505 None.

21506 **APPLICATION USAGE**

21507 None.

21508 **RATIONALE**

21509 None.

21510 **FUTURE DIRECTIONS**

21511 None.

21512 **SEE ALSO**21513 [csinh\(\)](#)21514 XBD [<complex.h>](#)21515 **CHANGE HISTORY**

21516 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**casinl()***System Interfaces*21517 **NAME**21518            **casinl** — complex arc sine functions21519 **SYNOPSIS**

21520            #include &lt;complex.h&gt;

21521            long double complex casinl(long double complex z);

21522 **DESCRIPTION**21523            Refer to *casin()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

21524 **NAME**

21525           catan, catanf, catanl — complex arc tangent functions

21526 **SYNOPSIS**

```
21527           #include <complex.h>
21528           double complex catan(double complex z);
21529           float complex catanf(float complex z);
21530           long double complex catanl(long double complex z);
```

21531 **DESCRIPTION**

21532 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21533 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21534 volume of POSIX.1-2008 defers to the ISO C standard.

21535           These functions shall compute the complex arc tangent of  $z$ , with branch cuts outside the  
 21536 interval  $[-i, +i]$  along the imaginary axis.

21537 **RETURN VALUE**

21538           These functions shall return the complex arc tangent value, in the range of a strip  
 21539 mathematically unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the  
 21540 real axis.

21541 **ERRORS**

21542           No errors are defined.

21543 **EXAMPLES**

21544           None.

21545 **APPLICATION USAGE**

21546           None.

21547 **RATIONALE**

21548           None.

21549 **FUTURE DIRECTIONS**

21550           None.

21551 **SEE ALSO**21552           [ctan\(\)](#)21553           XBD [<complex.h>](#)21554 **CHANGE HISTORY**

21555           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**catanh()**21556 **NAME**21557 `catanh, catanhf, catanhl` — complex arc hyperbolic tangent functions21558 **SYNOPSIS**

```
21559 #include <complex.h>
21560 double complex catanh(double complex z);
21561 float complex catanhf(float complex z);
21562 long double complex catanhl(long double complex z);
```

21563 **DESCRIPTION**

21564 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 21565 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21566 volume of POSIX.1-2008 defers to the ISO C standard.

21567 These functions shall compute the complex arc hyperbolic tangent of  $z$ , with branch cuts outside  
 21568 the interval  $[-1, +1]$  along the real axis.

21569 **RETURN VALUE**

21570 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip  
 21571 mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
 21572 imaginary axis.

21573 **ERRORS**

21574 No errors are defined.

21575 **EXAMPLES**

21576 None.

21577 **APPLICATION USAGE**

21578 None.

21579 **RATIONALE**

21580 None.

21581 **FUTURE DIRECTIONS**

21582 None.

21583 **SEE ALSO**21584 [ctanh\(\)](#)21585 XBD [<complex.h>](#)21586 **CHANGE HISTORY**

21587 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

*System Interfaces***catanl()**21588 **NAME**

21589         catanl — complex arc tangent functions

21590 **SYNOPSIS**

21591         #include &lt;complex.h&gt;

21592         long double complex catanl(long double complex z);

21593 **DESCRIPTION**21594         Refer to *catan()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**catclose()**21595 **NAME**21596 `catclose` — close a message catalog descriptor21597 **SYNOPSIS**21598 `#include <nl_types.h>`21599 `int catclose(nl_catd catd);`21600 **DESCRIPTION**21601 The `catclose()` function shall close the message catalog identified by `catd`. If a file descriptor is  
21602 used to implement the type `nl_catd`, that file descriptor shall be closed.21603 **RETURN VALUE**21604 Upon successful completion, `catclose()` shall return 0; otherwise, `-1` shall be returned, and `errno`  
21605 set to indicate the error.21606 **ERRORS**21607 The `catclose()` function may fail if:

21608 [EBADF] The catalog descriptor is not valid.

21609 [EINTR] The `catclose()` function was interrupted by a signal.21610 **EXAMPLES**

21611 None.

21612 **APPLICATION USAGE**

21613 None.

21614 **RATIONALE**

21615 None.

21616 **FUTURE DIRECTIONS**

21617 None.

21618 **SEE ALSO**21619 `catgets()`, `catopen()`21620 XBD `<nl_types.h>`21621 **CHANGE HISTORY**

21622 First released in Issue 2.

21623 **Issue 7**21624 The `catclose()` function is moved from the XSI option to the Base.

21625 **NAME**21626 `catgets` — read a program message21627 **SYNOPSIS**21628 `#include <nl_types.h>`21629 `char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);`21630 **DESCRIPTION**

21631 The `catgets()` function shall attempt to read message `msg_id`, in set `set_id`, from the message  
 21632 catalog identified by `catd`. The `catd` argument is a message catalog descriptor returned from an  
 21633 earlier call to `catopen()`. The results are undefined if `catd` is not a value returned by `catopen()` for  
 21634 a message catalog still open in the process. The `s` argument points to a default message string  
 21635 which shall be returned by `catgets()` if it cannot retrieve the identified message.

21636 The `catgets()` function need not be thread-safe.21637 **RETURN VALUE**

21638 If the identified message is retrieved successfully, `catgets()` shall return a pointer to an internal  
 21639 buffer area containing the null-terminated message string. If the call is unsuccessful for any  
 21640 reason, `s` shall be returned and `errno` shall be set to indicate the error.

21641 **ERRORS**21642 The `catgets()` function shall fail if:

21643 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 21644 was transferred.

21645 [ENOMSG] The message identified by `set_id` and `msg_id` is not in the message catalog.

21646 The `catgets()` function may fail if:

21647 [EBADF] The `catd` argument is not a valid message catalog descriptor open for reading.

21648 [EBADMSG] The message identified by `set_id` and `msg_id` in the specified message catalog  
 21649 did not satisfy implementation-defined security criteria.

21650 [EINVAL] The message catalog identified by `catd` is corrupted.

21651 **EXAMPLES**

21652 None.

21653 **APPLICATION USAGE**

21654 None.

21655 **RATIONALE**

21656 None.

21657 **FUTURE DIRECTIONS**

21658 None.

21659 **SEE ALSO**21660 `catclose()`, `catopen()`21661 XBD `<nl_types.h>`21662 **CHANGE HISTORY**

21663 First released in Issue 2.

**catgets()**21664 **Issue 5**

21665 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

21666 **Issue 6**

21667 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

21668 **Issue 7**

21669 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and  
21670 [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and  
21671 updating the DESCRIPTION to note that: “The results are undefined if *catd* is not a value  
21672 returned by *catopen()* for a message catalog still open in the process.

21673 Austin Group Interpretation 1003.1-2001 #148 is applied, adding

21674 The *catgets()* function is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

21675 **NAME**21676 `catopen` — open a message catalog21677 **SYNOPSIS**21678 `#include <nl_types.h>`21679 `nl_catd catopen(const char *name, int oflag);`21680 **DESCRIPTION**

21681 The `catopen()` function shall open a message catalog and return a message catalog descriptor.  
 21682 The `name` argument specifies the name of the message catalog to be opened. If `name` contains a  
 21683 `'/'`, then `name` specifies a complete name for the message catalog. Otherwise, the environment  
 21684 variable `NLSPATH` is used with `name` substituted for the `%N` conversion specification (see XBD  
 21685 Chapter 8, on page 173). If `NLSPATH` exists in the environment when the process starts, then if  
 21686 the process has appropriate privileges, the behavior of `catopen()` is undefined. If `NLSPATH` does  
 21687 not exist in the environment, or if a message catalog cannot be found in any of the components  
 21688 specified by `NLSPATH`, then an implementation-defined default path shall be used. This default  
 21689 may be affected by the setting of `LC_MESSAGES` if the value of `oflag` is `NL_CAT_LOCALE`, or  
 21690 the `LANG` environment variable if `oflag` is 0.

21691 A message catalog descriptor shall remain valid in a process until that process closes it, or a  
 21692 successful call to one of the `exec` functions. A change in the setting of the `LC_MESSAGES`  
 21693 category may invalidate existing open catalogs.

21694 If a file descriptor is used to implement message catalog descriptors, the `FD_CLOEXEC` flag  
 21695 shall be set; see `<fcntl.h>`.

21696 If the value of the `oflag` argument is 0, the `LANG` environment variable is used to locate the  
 21697 catalog without regard to the `LC_MESSAGES` category. If the `oflag` argument is  
 21698 `NL_CAT_LOCALE`, the `LC_MESSAGES` category is used to locate the message catalog (see XBD  
 21699 Section 8.2, on page 174).

21700 **RETURN VALUE**

21701 Upon successful completion, `catopen()` shall return a message catalog descriptor for use on  
 21702 subsequent calls to `catgets()` and `catclose()`. Otherwise, `catopen()` shall return `(nl_catd) -1` and set  
 21703 `errno` to indicate the error.

21704 **ERRORS**21705 The `catopen()` function may fail if:

21706 [EACCES] Search permission is denied for the component of the path prefix of the  
 21707 message catalog or read permission is denied for the message catalog.

21708 [EMFILE] All file descriptors available to the process are currently open.

21709 [ENAMETOOLONG]

21710 The length of a component of a pathname is longer than `{NAME_MAX}`.

21711 [ENAMETOOLONG]

21712 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a  
 21713 symbolic link produced an intermediate result with a length that exceeds  
 21714 `{PATH_MAX}`.

21715 [ENFILE] Too many files are currently open in the system.

21716 [ENOENT] The message catalog does not exist or the `name` argument points to an empty  
 21717 string.

**catopen()**

|       |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21718 | [ENOMEM]                 | Insufficient storage space is available.                                                                                                                                                                                                                                                                                                                                         |
| 21719 | [ENOTDIR]                | A component of the path prefix of the message catalog is not a directory, or the pathname of the message catalog contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory. |
| 21720 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21721 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21722 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21723 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21724 | <b>EXAMPLES</b>          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21725 | None.                    |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21726 | <b>APPLICATION USAGE</b> |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21727 |                          | Some implementations of <i>catopen()</i> use <i>malloc()</i> to allocate space for internal buffer areas. The <i>catopen()</i> function may fail if there is insufficient storage space available to accommodate these buffers.                                                                                                                                                  |
| 21728 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21729 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21730 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21731 |                          | Conforming applications must assume that message catalog descriptors are not valid after a call to one of the <i>exec</i> functions.                                                                                                                                                                                                                                             |
| 21732 |                          | Application developers should be aware that guidelines for the location of message catalogs have not yet been developed. Therefore they should take care to avoid conflicting with catalogs used by other applications and the standard utilities.                                                                                                                               |
| 21733 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21734 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21735 |                          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21735 | <b>RATIONALE</b>         |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21736 | None.                    |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21737 | <b>FUTURE DIRECTIONS</b> |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21738 | None.                    |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21739 | <b>SEE ALSO</b>          |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21740 |                          | <i>catclose()</i> , <i>catgets()</i>                                                                                                                                                                                                                                                                                                                                             |
| 21741 |                          | XBD Chapter 8 (on page 173), <code>&lt;fcntl.h&gt;</code> , <code>&lt;nl_types.h&gt;</code> ,                                                                                                                                                                                                                                                                                    |
| 21742 | <b>CHANGE HISTORY</b>    |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21743 |                          | First released in Issue 2.                                                                                                                                                                                                                                                                                                                                                       |
| 21744 | <b>Issue 7</b>           |                                                                                                                                                                                                                                                                                                                                                                                  |
| 21745 |                          | Austin Group Interpretation 1003.1-2001 #143 is applied.                                                                                                                                                                                                                                                                                                                         |
| 21746 |                          | SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.                                                                                                                                                                                                                                                                                                         |
| 21747 |                          | The <i>catopen()</i> function is moved from the XSI option to the Base.                                                                                                                                                                                                                                                                                                          |

21748 **NAME**

21749 cbrt, cbrtf, cbrtl — cube root functions

21750 **SYNOPSIS**

```
21751 #include <math.h>
21752 double cbrt(double x);
21753 float cbrtf(float x);
21754 long double cbrtl(long double x);
```

21755 **DESCRIPTION**

21756 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21757 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21758 volume of POSIX.1-2008 defers to the ISO C standard.

21759 These functions shall compute the real cube root of their argument  $x$ .21760 **RETURN VALUE**21761 Upon successful completion, these functions shall return the cube root of  $x$ .21762 MX If  $x$  is NaN, a NaN shall be returned.21763 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.21764 **ERRORS**

21765 No errors are defined.

21766 **EXAMPLES**

21767 None.

21768 **APPLICATION USAGE**

21769 None.

21770 **RATIONALE**

21771 For some applications, a true cube root function, which returns negative results for negative  
 21772 arguments, is more appropriate than  $\text{pow}(x, 1.0/3.0)$ , which returns a NaN for  $x$  less than 0.

21773 **FUTURE DIRECTIONS**

21774 None.

21775 **SEE ALSO**21776 XBD [<math.h>](#)21777 **CHANGE HISTORY**

21778 First released in Issue 4, Version 2.

21779 **Issue 5**

21780 Moved from X/OPEN UNIX extension to BASE.

21781 **Issue 6**21782 The `cbrt()` function is no longer marked as an extension.21783 The `cbrtf()` and `cbrtl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

21784 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 21785 revised to align with the ISO/IEC 9899:1999 standard.

21786 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 21787 marked.

**ccos()**21788 **NAME**21789 `ccos, ccosf, ccosl` — complex cosine functions21790 **SYNOPSIS**21791 `#include <complex.h>`21792 `double complex ccos(double complex z);`21793 `float complex ccosf(float complex z);`21794 `long double complex ccosl(long double complex z);`21795 **DESCRIPTION**21796 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21797 conflict between the requirements described here and the ISO C standard is unintentional. This  
21798 volume of POSIX.1-2008 defers to the ISO C standard.21799 These functions shall compute the complex cosine of *z*.21800 **RETURN VALUE**

21801 These functions shall return the complex cosine value.

21802 **ERRORS**

21803 No errors are defined.

21804 **EXAMPLES**

21805 None.

21806 **APPLICATION USAGE**

21807 None.

21808 **RATIONALE**

21809 None.

21810 **FUTURE DIRECTIONS**

21811 None.

21812 **SEE ALSO**21813 [\*ccos\(\)\*](#)21814 XBD [<complex.h>](#)21815 **CHANGE HISTORY**

21816 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21817 **NAME**

21818 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

21819 **SYNOPSIS**

21820 #include &lt;complex.h&gt;

21821 double complex ccosh(double complex z);

21822 float complex ccoshf(float complex z);

21823 long double complex ccoshl(long double complex z);

21824 **DESCRIPTION**

21825 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21826 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21827 volume of POSIX.1-2008 defers to the ISO C standard.

21828 These functions shall compute the complex hyperbolic cosine of *z*.21829 **RETURN VALUE**

21830 These functions shall return the complex hyperbolic cosine value.

21831 **ERRORS**

21832 No errors are defined.

21833 **EXAMPLES**

21834 None.

21835 **APPLICATION USAGE**

21836 None.

21837 **RATIONALE**

21838 None.

21839 **FUTURE DIRECTIONS**

21840 None.

21841 **SEE ALSO**21842 [cacosh\(\)](#)21843 XBD [<complex.h>](#)21844 **CHANGE HISTORY**

21845 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**ccosl()***System Interfaces*21846 **NAME**

21847           ccosl — complex cosine functions

21848 **SYNOPSIS**

21849           #include &lt;complex.h&gt;

21850           long double complex ccosl(long double complex z);

21851 **DESCRIPTION**21852           Refer to *ccos()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

21853 **NAME**

21854           ceil, ceilf, ceill — ceiling value function

21855 **SYNOPSIS**

```
21856       #include <math.h>
21857       double ceil(double x);
21858       float ceilf(float x);
21859       long double ceill(long double x);
```

21860 **DESCRIPTION**

21861 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
21862 conflict between the requirements described here and the ISO C standard is unintentional. This  
21863 volume of POSIX.1-2008 defers to the ISO C standard.

21864       These functions shall compute the smallest integral value not less than *x*.

21865       An application wishing to check for error situations should set *errno* to zero and call  
21866 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
21867 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
21868 zero, an error has occurred.

21869 **RETURN VALUE**

21870       Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not  
21871 less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

21872 MX       If *x* is NaN, a NaN shall be returned.

21873       If *x* is  $\pm 0$  or  $\pm \text{Inf}$ , *x* shall be returned.

21874 XSI       If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*  
21875 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

21876 **ERRORS**

21877       These functions shall fail if:

21878 XSI       Range Error       The result overflows.

21879       If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
21880 then *errno* shall be set to [ERANGE]. If the integer expression  
21881 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
21882 floating-point exception shall be raised.

21883 **EXAMPLES**

21884       None.

21885 **APPLICATION USAGE**

21886       The integral value returned by these functions need not be expressible as an **int** or **long**. The  
21887 return value should be tested before assigning it to an integer type to avoid the undefined  
21888 results of an integer overflow.

21889       The *ceil()* function can only overflow when the floating-point representation has  
21890 DBL\_MANT\_DIG > DBL\_MAX\_EXP.

21891       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
21892 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**ceil()**21893 **RATIONALE**

21894 None.

21895 **FUTURE DIRECTIONS**

21896 None.

21897 **SEE ALSO**21898 *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*21899 XBD Section 4.19 (on page 116), **<math.h>**21900 **CHANGE HISTORY**

21901 First released in Issue 1. Derived from Issue 1 of the SVID.

21902 **Issue 5**21903 The DESCRIPTION is updated to indicate how an application should check for an error. This  
21904 text was previously published in the APPLICATION USAGE section.21905 **Issue 6**21906 The *ceilf()* and *ceilll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.21907 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
21908 revised to align with the ISO/IEC 9899:1999 standard.21909 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
21910 marked.

21911 **NAME**

21912 cexp, cexpf, cexpl — complex exponential functions

21913 **SYNOPSIS**

21914 #include &lt;complex.h&gt;

21915 double complex cexp(double complex z);

21916 float complex cexpf(float complex z);

21917 long double complex cexpl(long double complex z);

21918 **DESCRIPTION**21919 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
21920 conflict between the requirements described here and the ISO C standard is unintentional. This  
21921 volume of POSIX.1-2008 defers to the ISO C standard.21922 These functions shall compute the complex exponent of  $z$ , defined as  $e^z$ .21923 **RETURN VALUE**21924 These functions shall return the complex exponential value of  $z$ .21925 **ERRORS**

21926 No errors are defined.

21927 **EXAMPLES**

21928 None.

21929 **APPLICATION USAGE**

21930 None.

21931 **RATIONALE**

21932 None.

21933 **FUTURE DIRECTIONS**

21934 None.

21935 **SEE ALSO**21936 [clog\(\)](#)21937 XBD [<complex.h>](#)21938 **CHANGE HISTORY**

21939 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**cfgetispeed()**21940 **NAME**21941 `cfgetispeed` — get input baud rate21942 **SYNOPSIS**21943 `#include <termios.h>`21944 `speed_t cfgetispeed(const struct termios *termios_p);`21945 **DESCRIPTION**21946 The `cfgetispeed()` function shall extract the input baud rate from the **termios** structure to which  
21947 the `termios_p` argument points.21948 This function shall return exactly the value in the **termios** data structure, without interpretation.21949 **RETURN VALUE**21950 Upon successful completion, `cfgetispeed()` shall return a value of type **speed\_t** representing the  
21951 input baud rate.21952 **ERRORS**

21953 No errors are defined.

21954 **EXAMPLES**

21955 None.

21956 **APPLICATION USAGE**

21957 None.

21958 **RATIONALE**21959 The term “baud” is used historically here, but is not technically correct. This is properly “bits per  
21960 second”, which may not be the same as baud. However, the term is used because of the  
21961 historical usage and understanding.21962 The `cfgetispeed()`, `cfgetispeed()`, `cfsetispeed()`, and `cfsetispeed()` functions do not take arguments as  
21963 numbers, but rather as symbolic names. There are two reasons for this:

- 21964 1. Historically, numbers were not used because of the way the rate was stored in the data  
21965 structure. This is retained even though a function is now used.
- 21966 2. More importantly, only a limited set of possible rates is at all portable, and this constrains  
21967 the application to that set.

21968 There is nothing to prevent an implementation accepting as an extension a number (such as 126),  
21969 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid  
21970 introducing ambiguity.21971 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications  
21972 in this volume of POSIX.1-2008 have made it possible to determine whether split rates are  
21973 supported and to support them without having to treat zero as a special case. Since this  
21974 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is  
21975 the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2008 does not  
21976 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit  
21977 from the two being equivalent. This volume of POSIX.1-2008 does not fully specify whether the  
21978 previous `cfsetispeed()` value is retained after a `tcgetattr()` as the actual value or as zero. Therefore,  
21979 conforming applications should always set both the input speed and output speed when setting  
21980 either.21981 In historical implementations, the baud rate information is traditionally kept in **c\_cflag**.  
21982 Applications should be written to presume that this might be the case (and thus not blindly copy  
21983 **c\_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c\_cflag**  
21984 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

21985 unused parts of the flag fields might be used by the implementation and should not be blindly  
21986 copied from the descriptions of one terminal device to another.

21987 **FUTURE DIRECTIONS**

21988 None.

21989 **SEE ALSO**

21990 [cfgetospeed\(\)](#), [cfsetispeed\(\)](#), [cfsetospeed\(\)](#), [tcgetattr\(\)](#)

21991 XBD Chapter 11 (on page 199), [<termios.h>](#)

21992 **CHANGE HISTORY**

21993 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**cfgetospeed()**

System Interfaces

21994 **NAME**21995 `cfgetospeed` — get output baud rate21996 **SYNOPSIS**21997 `#include <termios.h>`21998 `speed_t cfgetospeed(const struct termios *termios_p);`21999 **DESCRIPTION**22000 The `cfgetospeed()` function shall extract the output baud rate from the **termios** structure to which  
22001 the `termios_p` argument points.22002 This function shall return exactly the value in the **termios** data structure, without interpretation.22003 **RETURN VALUE**22004 Upon successful completion, `cfgetospeed()` shall return a value of type **speed\_t** representing the  
22005 output baud rate.22006 **ERRORS**

22007 No errors are defined.

22008 **EXAMPLES**

22009 None.

22010 **APPLICATION USAGE**

22011 None.

22012 **RATIONALE**22013 Refer to `cfgetispeed()`.22014 **FUTURE DIRECTIONS**

22015 None.

22016 **SEE ALSO**22017 `cfgetispeed()`, `cfsetispeed()`, `cfsetospeed()`, `tcgetattr()`22018 XBD Chapter 11 (on page 199); `<termios.h>`22019 **CHANGE HISTORY**

22020 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22021 **NAME**

22022 cfsetispeed — set input baud rate

22023 **SYNOPSIS**

22024 #include &lt;termios.h&gt;

22025 int cfsetispeed(struct termios \*termios\_p, speed\_t speed);

22026 **DESCRIPTION**22027 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by  
22028 *termios\_p* to *speed*.22029 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
22030 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
22031 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
22032 function is called.22033 **RETURN VALUE**22034 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and  
22035 *errno* may be set to indicate the error.22036 **ERRORS**22037 The *cfsetispeed()* function may fail if:22038 [EINVAL] The *speed* value is not a valid baud rate.22039 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
22040 <termios.h>.22041 **EXAMPLES**

22042 None.

22043 **APPLICATION USAGE**

22044 None.

22045 **RATIONALE**22046 Refer to *cfgetispeed()*.22047 **FUTURE DIRECTIONS**

22048 None.

22049 **SEE ALSO**22050 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*

22051 XBD Chapter 11 (on page 199), &lt;termios.h&gt;

22052 **CHANGE HISTORY**

22053 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22054 **Issue 6**22055 The following new requirements on POSIX implementations derive from alignment with the  
22056 Single UNIX Specification:

- 22057
- The optional setting of *errno* and the [EINVAL] error conditions are added.

**cfsetospeed()**

System Interfaces

22058 **NAME**

22059 cfsetospeed — set output baud rate

22060 **SYNOPSIS**

22061 #include &lt;termios.h&gt;

22062 int cfsetospeed(struct termios \*termios\_p, speed\_t speed);

22063 **DESCRIPTION**22064 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by  
22065 *termios\_p* to *speed*.22066 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
22067 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
22068 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
22069 function is called.22070 **RETURN VALUE**22071 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*  
22072 may be set to indicate the error.22073 **ERRORS**22074 The *cfsetospeed()* function may fail if:22075 [EINVAL] The *speed* value is not a valid baud rate.22076 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
22077 <termios.h>.22078 **EXAMPLES**

22079 None.

22080 **APPLICATION USAGE**

22081 None.

22082 **RATIONALE**22083 Refer to *cfgetispeed()*.22084 **FUTURE DIRECTIONS**

22085 None.

22086 **SEE ALSO**22087 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*

22088 XBD Chapter 11 (on page 199), &lt;termios.h&gt;

22089 **CHANGE HISTORY**

22090 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22091 **Issue 6**22092 The following new requirements on POSIX implementations derive from alignment with the  
22093 Single UNIX Specification:

- 22094
- The optional setting of *errno* and the [EINVAL] error conditions are added.

22095 **NAME**

22096 chdir — change working directory

22097 **SYNOPSIS**

```
22098 #include <unistd.h>
22099 int chdir(const char *path);
```

22100 **DESCRIPTION**

22101 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*  
 22102 argument to become the current working directory; that is, the starting point for path searches  
 22103 for pathnames not beginning with *'/'*.

22104 **RETURN VALUE**

22105 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current  
 22106 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

22107 **ERRORS**

22108 The *chdir()* function shall fail if:

- 22109 [EACCES] Search permission is denied for any component of the pathname.
- 22110 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 22111 argument.
- 22112 [ENAMETOOLONG]  
 22113 The length of a component of a pathname is longer than {NAME\_MAX}.
- 22114 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty  
 22115 string.
- 22116 [ENOTDIR] A component of the pathname is not a directory.

22117 The *chdir()* function may fail if:

- 22118 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 22119 resolution of the *path* argument.
- 22120 [ENAMETOOLONG]  
 22121 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 22122 symbolic link produced an intermediate result with a length that exceeds  
 22123 {PATH\_MAX}.

22124 **EXAMPLES**22125 **Changing the Current Working Directory**

22126 The following example makes the value pointed to by **directory**, */tmp*, the current working  
 22127 directory.

```
22128 #include <unistd.h>
22129 ...
22130 char *directory = "/tmp";
22131 int ret;
22132 ret = chdir (directory);
```

**chdir()**22133 **APPLICATION USAGE**

22134 None.

22135 **RATIONALE**22136 The *chdir()* function only affects the working directory of the current process.22137 **FUTURE DIRECTIONS**

22138 None.

22139 **SEE ALSO**22140 *getcwd()*22141 XBD <*unistd.h*>22142 **CHANGE HISTORY**

22143 First released in Issue 1. Derived from Issue 1 of the SVID.

22144 **Issue 6**

22145 The APPLICATION USAGE section is added.

22146 The following new requirements on POSIX implementations derive from alignment with the  
22147 Single UNIX Specification:

- 22148 • The [ELOOP] mandatory error condition is added.
- 22149 • A second [ENAMETOOLONG] is added as an optional error condition.

22150 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22151 • The [ELOOP] optional error condition is added.

22152 **Issue 7**

22153 Austin Group Interpretation 1003.1-2001 #143 is applied.

22154 **NAME**

22155 chmod, fchmodat — change mode of a file relative to directory file descriptor

22156 **SYNOPSIS**

22157 #include &lt;sys/stat.h&gt;

22158 int chmod(const char \*path, mode\_t mode);

22159 int fchmodat(int fd, const char \*path, mode\_t mode, int flag);

22160 **DESCRIPTION**

22161 XSI The *chmod()* function shall change S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits of  
 22162 the file named by the pathname pointed to by the *path* argument to the corresponding bits in the  
 22163 *mode* argument. The application shall ensure that the effective user ID of the process matches the  
 22164 owner of the file or the process has appropriate privileges in order to do this.

22165 XSI S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits are described in <sys/stat.h>.

22166 If the calling process does not have appropriate privileges, and if the group ID of the file does  
 22167 not match the effective group ID or one of the supplementary group IDs and if the file is a  
 22168 regular file, bit S\_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon  
 22169 successful return from *chmod()*.

22170 Additional implementation-defined restrictions may cause the S\_ISUID and S\_ISGID bits in  
 22171 *mode* to be ignored.

22172 The effect on file descriptors for files open at the time of a call to *chmod()* is implementation-  
 22173 defined.

22174 Upon successful completion, *chmod()* shall mark for update the last file status change timestamp  
 22175 of the file.

22176 The *fchmodat()* function shall be equivalent to the *chmod()* function except in the case where *path*  
 22177 specifies a relative path. In this case the file to be changed is determined relative to the directory  
 22178 associated with the file descriptor *fd* instead of the current working directory. If the file  
 22179 descriptor was opened without O\_SEARCH, the function shall check whether directory searches  
 22180 are permitted using the current permissions of the directory underlying the file descriptor. If the  
 22181 file descriptor was opened with O\_SEARCH, the function shall not perform the check.

22182 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 22183 in <fcntl.h>:

22184 AT\_SYMLINK\_NOFOLLOW

22185 If *path* names a symbolic link, then the mode of the symbolic link is changed.

22186 If *fchmodat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 22187 directory is used. If also *flag* is zero, the behavior shall be identical to a call to *chmod()*.

22188 **RETURN VALUE**

22189 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 22190 return -1 and set *errno* to indicate the error. If -1 is returned, no change to the file mode occurs.

22191 **ERRORS**

22192 These functions shall fail if:

22193 [EACCES] Search permission is denied on a component of the path prefix.

22194 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 22195 argument.

**chmod()**

|       |                |                                                                                                    |
|-------|----------------|----------------------------------------------------------------------------------------------------|
| 22196 | [ENAMETOOLONG] |                                                                                                    |
| 22197 |                | The length of a component of a pathname is longer than {NAME_MAX}.                                 |
| 22198 | [ENOENT]       | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.       |
| 22199 | [ENOTDIR]      | A component of the path prefix is not a directory, or the <i>path</i> argument                     |
| 22200 |                | contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing |
| 22201 |                | <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file       |
| 22202 |                | that is neither a directory nor a symbolic link to a directory.                                    |
| 22203 | [EPERM]        | The effective user ID does not match the owner of the file and the process does                    |
| 22204 |                | not have appropriate privileges.                                                                   |
| 22205 | [EROFS]        | The named file resides on a read-only file system.                                                 |
| 22206 |                | The <i>fchmodat()</i> function shall fail if:                                                      |
| 22207 | [EACCES]       | <i>fd</i> was not opened with O_SEARCH and the permissions of the directory                        |
| 22208 |                | underlying <i>fd</i> do not permit directory searches.                                             |
| 22209 | [EBADF]        | The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is           |
| 22210 |                | neither AT_FDCWD nor a valid file descriptor open for reading or searching.                        |
| 22211 |                | These functions may fail if:                                                                       |
| 22212 | [EINTR]        | A signal was caught during execution of the function.                                              |
| 22213 | [EINVAL]       | The value of the <i>mode</i> argument is invalid.                                                  |
| 22214 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during                                     |
| 22215 |                | resolution of the <i>path</i> argument.                                                            |
| 22216 | [ENAMETOOLONG] |                                                                                                    |
| 22217 |                | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a                           |
| 22218 |                | symbolic link produced an intermediate result with a length that exceeds                           |
| 22219 |                | {PATH_MAX}.                                                                                        |
| 22220 |                | The <i>fchmodat()</i> function may fail if:                                                        |
| 22221 | [EINVAL]       | The value of the <i>flag</i> argument is invalid.                                                  |
| 22222 | [ENOTDIR]      | The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a           |
| 22223 |                | file descriptor associated with a directory.                                                       |
| 22224 | [EOPNOTSUPP]   | The AT_SYMLINK_NOFOLLOW bit is set in the <i>flag</i> argument, <i>path</i> names a                |
| 22225 |                | symbolic link, and the system does not support changing the mode of a                              |
| 22226 |                | symbolic link.                                                                                     |

22227 **EXAMPLES**22228 **Setting Read Permissions for User, Group, and Others**

22229 The following example sets read permissions for the owner, group, and others.

```

22230 #include <sys/stat.h>
22231 const char *path;
22232 ...
22233 chmod(path, S_IRUSR|S_IRGRP|S_IROTH);

```

### 22234 **Setting Read, Write, and Execute Permissions for the Owner Only**

22235 The following example sets read, write, and execute permissions for the owner, and no  
22236 permissions for group and others.

```
22237 #include <sys/stat.h>
22238 const char *path;
22239 ...
22240 chmod(path, S_IRWXU);
```

### 22241 **Setting Different Permissions for Owner, Group, and Other**

22242 The following example sets owner permissions for CHANGEFILE to read, write, and execute,  
22243 group permissions to read and execute, and other permissions to read.

```
22244 #include <sys/stat.h>
22245 #define CHANGEFILE "/etc/myfile"
22246 ...
22247 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

### 22248 **Setting and Checking File Permissions**

22249 The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls  
22250 the `stat()` function to verify the permissions.

```
22251 #include <sys/types.h>
22252 #include <sys/stat.h>
22253 int status;
22254 struct stat buffer
22255 ...
22256 chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
22257 status = stat("home/cnd/mod1", &buffer);
```

### 22258 **APPLICATION USAGE**

22259 In order to ensure that the `S_ISUID` and `S_ISGID` bits are set, an application requiring this  
22260 should use `stat()` after a successful `chmod()` to verify this.

22261 Any file descriptors currently open by any process on the file could possibly become invalid if  
22262 the mode of the file is changed to a value which would deny access to that process. One  
22263 situation where this could occur is on a stateless file system. This behavior will not occur in a  
22264 conforming environment.

### 22265 **RATIONALE**

22266 This volume of POSIX.1-2008 specifies that the `S_ISGID` bit is cleared by `chmod()` on a regular file  
22267 under certain conditions. This is specified on the assumption that regular files may be executed,  
22268 and the system should prevent users from making executable `setgid()` files perform with  
22269 privileges that the caller does not have. On implementations that support execution of other file  
22270 types, the `S_ISGID` bit should be cleared for those file types under the same circumstances.

22271 Implementations that use the `S_ISUID` bit to indicate some other function (for example,  
22272 mandatory record locking) on non-executable files need not clear this bit on writing. They  
22273 should clear the bit for executable files and any other cases where the bit grants special powers  
22274 to processes that change the file contents. Similar comments apply to the `S_ISGID` bit.

22275 The purpose of the `fchmodat()` function is to enable changing the mode of files in directories

**chmod()**

22276 other than the current working directory without exposure to race conditions. Any part of the  
 22277 path of a file could be changed in parallel to a call to *chmod()*, resulting in unspecified behavior.  
 22278 By opening a file descriptor for the target directory and using the *fchmodat()* function it can be  
 22279 guaranteed that the changed file is located relative to the desired directory. Some  
 22280 implementations might allow changing the mode of symbolic links. This is not supported by the  
 22281 interfaces in the POSIX specification. Systems with such support provide an interface named  
 22282 *lchmod()*. To support such implementations *fchmodat()* has a *flag* parameter.

22283 **FUTURE DIRECTIONS**

22284 None.

22285 **SEE ALSO**22286 *access()*, *chown()*, *exec*, *fstatat()*, *fstatvfs()*, *mkdir()*, *mkfifo()*, *mknod()*, *open()*22287 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)22288 **CHANGE HISTORY**

22289 First released in Issue 1. Derived from Issue 1 of the SVID.

22290 **Issue 6**22291 The following new requirements on POSIX implementations derive from alignment with the  
 22292 Single UNIX Specification:

- 22293 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 22294 required for conforming implementations of previous POSIX specifications, it was not  
 22295 required for UNIX applications.
- 22296 • The [EINVAL] and [EINTR] optional error conditions are added.
- 22297 • A second [ENAMETOOLONG] is added as an optional error condition.

22298 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22299 • The [ELOOP] optional error condition is added.

22300 The normative text is updated to avoid use of the term “must” for application requirements.

22301 **Issue 7**

22302 Austin Group Interpretation 1003.1-2001 #143 is applied.

22303 The *fchmodat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 22304 Set Part 2.

22305 Changes are made related to support for finegrained timestamps.

22306 Changes are made to allow a directory to be opened for searching.

22307 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 22308 pathname exists but is not a directory or a symbolic link to a directory.

22309 **NAME**

22310 chown, fchownat — change owner and group of a file relative to directory file descriptor

22311 **SYNOPSIS**

```
22312 #include <unistd.h>
22313
22313 int chown(const char *path, uid_t owner, gid_t group);
22314 int fchownat(int fd, const char *path, uid_t owner, gid_t group,
22315             int flag);
```

22316 **DESCRIPTION**

22317 The *chown()* function shall change the user and group ownership of a file.

22318 The *path* argument points to a pathname naming a file. The user ID and group ID of the named  
22319 file shall be set to the numeric values contained in *owner* and *group*, respectively.

22320 Only processes with an effective user ID equal to the user ID of the file or with appropriate  
22321 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for  
22322 *path*:

- 22323 • Changing the user ID is restricted to processes with appropriate privileges.
- 22324 • Changing the group ID is permitted to a process with an effective user ID equal to the user  
22325 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's  
22326 user ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to  
22327 one of its supplementary group IDs.

22328 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of  
22329 the file mode are set, and the process does not have appropriate privileges, the set-user-ID  
22330 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful  
22331 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,  
22332 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is  
22333 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*  
22334 function is successfully invoked on a file that is not a regular file and one or more of the  
22335 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID  
22336 bits may be cleared.

22337 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the  
22338 file shall not be changed. If both *owner* and *group* are `-1`, the times need not be updated.

22339 Upon successful completion, *chown()* shall mark for update the last file status change timestamp  
22340 of the file.

22341 The *fchownat()* function shall be equivalent to the *chown()* and *lchown()* functions except in the  
22342 case where *path* specifies a relative path. In this case the file to be changed is determined relative  
22343 to the directory associated with the file descriptor *fd* instead of the current working directory. If  
22344 the file descriptor was opened without `O_SEARCH`, the function shall check whether directory  
22345 searches are permitted using the current permissions of the directory underlying the file  
22346 descriptor. If the file descriptor was opened with `O_SEARCH`, the function shall not perform the  
22347 check.

22348 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
22349 in `<fcntl.h>`:

22350 `AT_SYMLINK_NOFOLLOW`

22351 If *path* names a symbolic link, ownership of the symbolic link is changed.

22352 If *fchownat()* is passed the special value `AT_FDCWD` in the *fd* parameter, the current working  
22353 directory is used and the behavior shall be identical to a call to *chown()* or *lchown()* respectively,

**chown()**

- 22354 depending on whether or not the `AT_SYMLINK_NOFOLLOW` bit is set in the *flag* argument.
- 22355 **RETURN VALUE**
- 22356 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
22357 return `-1` and set *errno* to indicate the error. If `-1` is returned, no changes are made in the user ID  
22358 and group ID of the file.
- 22359 **ERRORS**
- 22360 These functions shall fail if:
- 22361 `[EACCES]` Search permission is denied on a component of the path prefix.
- 22362 `[ELOOP]` A loop exists in symbolic links encountered during resolution of the *path*  
22363 argument.
- 22364 `[ENAMETOOLONG]`  
22365 The length of a component of a pathname is longer than `{NAME_MAX}`.
- 22366 `[ENOENT]` A component of *path* does not name an existing file or *path* is an empty string.
- 22367 `[ENOTDIR]` A component of the path prefix is not a directory, or the *path* argument  
22368 contains at least one non-`<slash>` character and ends with one or more trailing  
22369 `<slash>` characters and the last pathname component names an existing file  
22370 that is neither a directory nor a symbolic link to a directory.
- 22371 `[EPERM]` The effective user ID does not match the owner of the file, or the calling  
22372 process does not have appropriate privileges and  
22373 `_POSIX_CHOWN_RESTRICTED` indicates that such privilege is required.
- 22374 `[EROFS]` The named file resides on a read-only file system.
- 22375 The *fchownat()* function shall fail if:
- 22376 `[EACCES]` *fd* was not opened with `O_SEARCH` and the permissions of the directory  
22377 underlying *fd* do not permit directory searches.
- 22378 `[EBADF]` The *path* argument does not specify an absolute path and the *fd* argument is  
22379 neither `AT_FDCWD` nor a valid file descriptor open for reading or searching.
- 22380 These functions may fail if:
- 22381 `[EIO]` An I/O error occurred while reading or writing to the file system.
- 22382 `[EINTR]` The *chown()* function was interrupted by a signal which was caught.
- 22383 `[EINVAL]` The owner or group ID supplied is not a value supported by the  
22384 implementation.
- 22385 `[ELOOP]` More than `{SYMLOOP_MAX}` symbolic links were encountered during  
22386 resolution of the *path* argument.
- 22387 `[ENAMETOOLONG]`  
22388 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a  
22389 symbolic link produced an intermediate result with a length that exceeds  
22390 `{PATH_MAX}`.
- 22391 The *fchownat()* function may fail if:
- 22392 `[EINVAL]` The value of the *flag* argument is not valid.

22393 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
 22394 file descriptor associated with a directory.

#### 22395 EXAMPLES

22396 None.

#### 22397 APPLICATION USAGE

22398 Although *chown()* can be used on some implementations by the file owner to change the owner  
 22399 and group to any desired values, the only portable use of this function is to change the group of  
 22400 a file to the effective GID of the calling process or to a member of its group set.

#### 22401 RATIONALE

22402 System III and System V allow a user to give away files; that is, the owner of a file may change  
 22403 its user ID to anything. This is a serious problem for implementations that are intended to meet  
 22404 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the  
 22405 user ID of a file. Some government agencies (usually not ones concerned directly with security)  
 22406 find this limitation too confining. This volume of POSIX.1-2008 uses *may* to permit secure  
 22407 implementations while not disallowing System V.

22408 System III and System V allow the owner of a file to change the group ID to anything. Version 7  
 22409 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to  
 22410 change the group ID of a file to its effective group ID or to any of the groups in the list of  
 22411 supplementary group IDs, but to no others.

22412 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate  
 22413 privileged process clear the S\_ISGID and the S\_ISUID bits for regular files, and permits them to  
 22414 be cleared for other types of files. This is so that changes in accessibility do not accidentally  
 22415 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-  
 22416 executable data files also clears the mandatory file locking bit (shared with S\_ISGID), which is  
 22417 an extension on many implementations (it first appeared in System V). These bits should only be  
 22418 required to be cleared on regular files that have one or more of their execute bits set.

22419 The purpose of the *fchownat()* function is to enable changing ownership of files in directories  
 22420 other than the current working directory without exposure to race conditions. Any part of the  
 22421 path of a file could be changed in parallel to a call to *chown()* or *lchown()*, resulting in  
 22422 unspecified behavior. By opening a file descriptor for the target directory and using the  
 22423 *fchownat()* function it can be guaranteed that the changed file is located relative to the desired  
 22424 directory.

#### 22425 FUTURE DIRECTIONS

22426 None.

#### 22427 SEE ALSO

22428 *chmod()*, *fpathconf()*, *lchown()*

22429 XBD <fcntl.h>, <sys/types.h>, <unistd.h>

#### 22430 CHANGE HISTORY

22431 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 22432 Issue 6

22433 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 22434 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is
- 22435 restored.

**chown()**

- 22436 • The [EPERM] error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.  
 22437 This is since its operand is a pathname and applications should be aware that the error  
 22438 may not occur for that pathname if the file system does not support  
 22439 `_POSIX_CHOWN_RESTRICTED`.
- 22440 The following new requirements on POSIX implementations derive from alignment with the  
 22441 Single UNIX Specification:
- 22442 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 22443 required for conforming implementations of previous POSIX specifications, it was not  
 22444 required for UNIX applications.
  - 22445 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the  
 22446 group ID only. A corresponding change is made for group.
  - 22447 • The [ELOOP] mandatory error condition is added.
  - 22448 • The [EIO] and [EINTR] optional error conditions are added.
  - 22449 • A second [ENAMETOOLONG] is added as an optional error condition.
- 22450 The following changes were made to align with the IEEE P1003.1a draft standard:
- 22451 • Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when  
 22452 the process has appropriate privileges.
  - 22453 • The [ELOOP] optional error condition is added.
- 22454 **Issue 7**
- 22455 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 22456 The `fchownat()` function is added from The Open Group Technical Standard, 2006, Extended API  
 22457 Set Part 2.
- 22458 Changes are made related to support for finegrained timestamps.
- 22459 Changes are made to allow a directory to be opened for searching.
- 22460 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 22461 pathname exists but is not a directory or a symbolic link to a directory.

22462 **NAME**

22463           cimag, cimagf, cimagl — complex imaginary functions

22464 **SYNOPSIS**

22465           #include &lt;complex.h&gt;

22466           double cimag(double complex z);

22467           float cimagf(float complex z);

22468           long double cimagl(long double complex z);

22469 **DESCRIPTION**22470 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
22471 conflict between the requirements described here and the ISO C standard is unintentional. This  
22472 volume of POSIX.1-2008 defers to the ISO C standard.

22473           These functions shall compute the imaginary part of z.

22474 **RETURN VALUE**

22475           These functions shall return the imaginary part value (as a real).

22476 **ERRORS**

22477           No errors are defined.

22478 **EXAMPLES**

22479           None.

22480 **APPLICATION USAGE**

22481           For a variable z of complex type:

22482           z == creal(z) + cimag(z)\*I

22483 **RATIONALE**

22484           None.

22485 **FUTURE DIRECTIONS**

22486           None.

22487 **SEE ALSO**

22488           carg(), conj(), cproj(), creal()

22489           XBD &lt;complex.h&gt;

22490 **CHANGE HISTORY**

22491           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**clearerr()**22492 **NAME**

22493 clearerr — clear indicators on a stream

22494 **SYNOPSIS**

22495 #include &lt;stdio.h&gt;

22496 void clearerr(FILE \*stream);

22497 **DESCRIPTION**22498 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22499 conflict between the requirements described here and the ISO C standard is unintentional. This  
22500 volume of POSIX.1-2008 defers to the ISO C standard.22501 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which  
22502 *stream* points.22503 **RETURN VALUE**22504 The *clearerr()* function shall not return a value.22505 **ERRORS**

22506 No errors are defined.

22507 **EXAMPLES**

22508 None.

22509 **APPLICATION USAGE**

22510 None.

22511 **RATIONALE**

22512 None.

22513 **FUTURE DIRECTIONS**

22514 None.

22515 **SEE ALSO**

22516 XBD &lt;stdio.h&gt;

22517 **CHANGE HISTORY**

22518 First released in Issue 1. Derived from Issue 1 of the SVID.

22519 **NAME**

22520 clock — report CPU time used

22521 **SYNOPSIS**

22522 #include &lt;time.h&gt;

22523 clock\_t clock(void);

22524 **DESCRIPTION**

22525 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22526 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22527 volume of POSIX.1-2008 defers to the ISO C standard.

22528 The *clock()* function shall return the implementation's best approximation to the processor time  
 22529 used by the process since the beginning of an implementation-defined era related only to the  
 22530 process invocation.

22531 **RETURN VALUE**

22532 To determine the time in seconds, the value returned by *clock()* should be divided by the value  
 22533 XSI of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.  
 22534 If the processor time used is not available or its value cannot be represented, the function shall  
 22535 return the value `(clock_t)-1`.

22536 **ERRORS**

22537 No errors are defined.

22538 **EXAMPLES**

22539 None.

22540 **APPLICATION USAGE**

22541 In order to measure the time spent in a program, *clock()* should be called at the start of the  
 22542 program and its return value subtracted from the value returned by subsequent calls. The value  
 22543 returned by *clock()* is defined for compatibility across systems that have clocks with different  
 22544 resolutions. The resolution on any particular system need not be to microsecond accuracy.

22545 The value returned by *clock()* may wrap around on some implementations. For example, on a  
 22546 machine with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

22547 **RATIONALE**

22548 None.

22549 **FUTURE DIRECTIONS**

22550 None.

22551 **SEE ALSO**22552 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*22553 XBD `<time.h>`22554 **CHANGE HISTORY**

22555 First released in Issue 1. Derived from Issue 1 of the SVID.

**clock\_getcpuclockid()**22556 **NAME**22557 clock\_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)22558 **SYNOPSIS**

```
22559 CPT #include <time.h>
22560 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

22561 **DESCRIPTION**

22562 The *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process  
 22563 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the  
 22564 clock ID of this clock shall be returned in *clock\_id*.

22565 If *pid* is zero, the *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of  
 22566 the process making the call, in *clock\_id*.

22567 The conditions under which one process has permission to obtain the CPU-time clock ID of  
 22568 other processes are implementation-defined.

22569 **RETURN VALUE**

22570 Upon successful completion, *clock\_getcpuclockid()* shall return zero; otherwise, an error number  
 22571 shall be returned to indicate the error.

22572 **ERRORS**

22573 The *clock\_getcpuclockid()* function shall fail if:

22574 [EPERM] The requesting process does not have permission to access the CPU-time clock  
 22575 for the process.

22576 The *clock\_getcpuclockid()* function may fail if:

22577 [ESRCH] No process can be found corresponding to the process specified by *pid*.

22578 **EXAMPLES**

22579 None.

22580 **APPLICATION USAGE**

22581 The *clock\_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not be  
 22582 provided on all implementations.

22583 **RATIONALE**

22584 None.

22585 **FUTURE DIRECTIONS**

22586 None.

22587 **SEE ALSO**

22588 [clock\\_getres\(\)](#), [timer\\_create\(\)](#)

22589 XBD [<time.h>](#)

22590 **CHANGE HISTORY**

22591 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

22592 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

22593 **NAME**

22594 clock\_getres, clock\_gettime, clock\_settime — clock and timer functions

22595 **SYNOPSIS**

```

22596 CX #include <time.h>
22597
22597 int clock_getres(clockid_t clock_id, struct timespec *res);
22598 int clock_gettime(clockid_t clock_id, struct timespec *tp);
22599 int clock_settime(clockid_t clock_id, const struct timespec *tp);

```

22600 **DESCRIPTION**

22601 The *clock\_getres()* function shall return the resolution of any clock. Clock resolutions are  
 22602 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the  
 22603 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,  
 22604 the clock resolution is not returned. If the *time* argument of *clock\_settime()* is not a multiple of *res*,  
 22605 then the value is truncated to a multiple of *res*.

22606 The *clock\_gettime()* function shall return the current value *tp* for the specified clock, *clock\_id*.

22607 The *clock\_settime()* function shall set the specified clock, *clock\_id*, to the value specified by *tp*.  
 22608 Time values that are between two consecutive non-negative integer multiples of the resolution of  
 22609 the specified clock shall be truncated down to the smaller multiple of the resolution.

22610 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that  
 22611 is meaningful only within a process). All implementations shall support a *clock\_id* of  
 22612 CLOCK\_REALTIME as defined in <time.h>. This clock represents the clock measuring real time  
 22613 for the system. For this clock, the values returned by *clock\_gettime()* and specified by  
 22614 *clock\_settime()* represent the amount of time (in seconds and nanoseconds) since the Epoch. An  
 22615 implementation may also support additional clocks. The interpretation of time values for these  
 22616 clocks is unspecified.

22617 If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 22618 shall be used to determine the time of expiration for absolute time services based upon the  
 22619 CLOCK\_REALTIME clock. This applies to the time at which armed absolute timers expire. If the  
 22620 absolute time requested at the invocation of such a time service is before the new value of the  
 22621 clock, the time service shall expire immediately as if the clock had reached the requested time  
 22622 normally.

22623 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on  
 22624 threads that are blocked waiting for a relative time service based upon this clock, including the  
 22625 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.  
 22626 Consequently, these time services shall expire when the requested relative interval elapses,  
 22627 independently of the new or old value of the clock.

22628 MON If the Monotonic Clock option is supported, all implementations shall support a *clock\_id* of  
 22629 CLOCK\_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the  
 22630 system. For this clock, the value returned by *clock\_gettime()* represents the amount of time (in  
 22631 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up  
 22632 time, or the Epoch). This point does not change after system start-up time. The value of the  
 22633 CLOCK\_MONOTONIC clock cannot be set via *clock\_settime()*. This function shall fail if it is  
 22634 invoked with a *clock\_id* argument of CLOCK\_MONOTONIC.

22635 The effect of setting a clock via *clock\_settime()* on armed per-process timers associated with a  
 22636 clock other than CLOCK\_REALTIME is implementation-defined.

22637 If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 22638 shall be used to determine the time at which the system shall awaken a thread blocked on an

**clock\_getres()**

- 22639 absolute *clock\_nanosleep()* call based upon the CLOCK\_REALTIME clock. If the absolute time  
 22640 requested at the invocation of such a time service is before the new value of the clock, the call  
 22641 shall return immediately as if the clock had reached the requested time normally.
- 22642 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on any  
 22643 thread that is blocked on a relative *clock\_nanosleep()* call. Consequently, the call shall return  
 22644 when the requested relative interval elapses, independently of the new or old value of the clock.
- 22645 Appropriate privileges to set a particular clock are implementation-defined.
- 22646 CPT If \_POSIX\_CPUTIME is defined, implementations shall support clock ID values obtained by  
 22647 invoking *clock\_getcpuclockid()*, which represent the CPU-time clock of a given process.  
 22648 Implementations shall also support the special **clockid\_t** value  
 22649 CLOCK\_PROCESS\_CPUTIME\_ID, which represents the CPU-time clock of the calling process  
 22650 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 22651 returned by *clock\_gettime()* and specified by *clock\_settime()* represent the amount of execution  
 22652 time of the process associated with the clock. Changing the value of a CPU-time clock via  
 22653 *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see  
 22654 [Scheduling Policies](#), on page 501).
- 22655 TCT If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support clock ID values  
 22656 obtained by invoking *pthread\_getcpuclockid()*, which represent the CPU-time clock of a given  
 22657 thread. Implementations shall also support the special **clockid\_t** value  
 22658 CLOCK\_THREAD\_CPUTIME\_ID, which represents the CPU-time clock of the calling thread  
 22659 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 22660 returned by *clock\_gettime()* and specified by *clock\_settime()* shall represent the amount of  
 22661 execution time of the thread associated with the clock. Changing the value of a CPU-time clock  
 22662 via *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy  
 22663 (see [Scheduling Policies](#), on page 501).
- 22664 **RETURN VALUE**
- 22665 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that  
 22666 an error occurred, and *errno* shall be set to indicate the error.
- 22667 **ERRORS**
- 22668 The *clock\_getres()*, *clock\_gettime()*, and *clock\_settime()* functions shall fail if:
- 22669 [EINVAL] The *clock\_id* argument does not specify a known clock.
- 22670 The *clock\_settime()* function shall fail if:
- 22671 [EINVAL] The *tp* argument to *clock\_settime()* is outside the range for the given clock ID.
- 22672 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or  
 22673 equal to 1 000 million.
- 22674 MON [EINVAL] The value of the *clock\_id* argument is CLOCK\_MONOTONIC.
- 22675 The *clock\_settime()* function may fail if:
- 22676 [EPERM] The requesting process does not have appropriate privileges to set the  
 22677 specified clock.

22678 **EXAMPLES**

22679 None.

22680 **APPLICATION USAGE**

22681 Note that the absolute value of the monotonic clock is meaningless (because its origin is  
 22682 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the  
 22683 fact that the value of this clock is never set and, therefore, that time intervals measured with this  
 22684 clock will not be affected by calls to *clock\_settime()*.

22685 **RATIONALE**

22686 None.

22687 **FUTURE DIRECTIONS**

22688 None.

22689 **SEE ALSO**

22690 *clock\_getcpuclockid()*, *clock\_nanosleep()*, *ctime()*, *mq\_receive()*, *mq\_send()*, *nanosleep()*,  
 22691 *pthread\_mutex\_timedlock()*, *sem\_timedwait()*, *time()*, *timer\_create()*, *timer\_getoverrun()*

22692 XBD <[time.h](#)>22693 **CHANGE HISTORY**

22694 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

22695 **Issue 6**

22696 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 22697 implementation does not support the Timers option.

22698 The APPLICATION USAGE section is added.

22699 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22700 • Clarification is added of the effect of resetting the clock resolution.

22701 CPU-time clocks and the *clock\_getcpuclockid()* function are added for alignment with IEEE Std  
 22702 1003.1d-1999.

22703 The following changes are added for alignment with IEEE Std 1003.1j-2000:

- 22704 • The DESCRIPTION is updated as follows:

- 22705 — The value returned by *clock\_gettime()* for CLOCK\_MONOTONIC is specified.

- 22706 — The *clock\_settime()* function failing for CLOCK\_MONOTONIC is specified.

- 22707 — The effects of *clock\_settime()* on the *clock\_nanosleep()* function with respect to  
 22708 CLOCK\_REALTIME are specified.

- 22709 • An [EINVAL] error is added to the ERRORS section, indicating that *clock\_settime()* fails for  
 22710 CLOCK\_MONOTONIC.

- 22711 • The APPLICATION USAGE section notes that the CLOCK\_MONOTONIC clock need not  
 22712 and shall not be set by *clock\_settime()* since the absolute value of the  
 22713 CLOCK\_MONOTONIC clock is meaningless.

- 22714 • The *clock\_nanosleep()*, *mq\_timedreceive()*, *mq\_timedsend()*, *pthread\_mutex\_timedlock()*,  
 22715 *sem\_timedwait()*, *timer\_create()*, and *timer\_settime()* functions are added to the SEE ALSO  
 22716 section.

**clock\_getres()**22717 **Issue 7**

22718           Functionality relating to the Clock Selection option is moved to the Base.

22719           The *clock\_getres()*, *clock\_gettime()*, and *clock\_settime()* functions are moved from the Timers  
22720           option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

22721 **NAME**22722 `clock_nanosleep` — high resolution sleep with specifiable clock22723 **SYNOPSIS**

```
22724 CX #include <time.h>
22725 int clock_nanosleep(clockid_t clock_id, int flags,
22726 const struct timespec *rqtp, struct timespec *rmtp);
```

22727 **DESCRIPTION**

22728 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall  
 22729 cause the current thread to be suspended from execution until either the time interval specified  
 22730 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to  
 22731 invoke a signal-catching function, or the process is terminated. The clock used to measure the  
 22732 time shall be the clock specified by *clock\_id*.

22733 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall  
 22734 cause the current thread to be suspended from execution until either the time value of the clock  
 22735 specified by *clock\_id* reaches the absolute time specified by the *rqtp* argument, or a signal is  
 22736 delivered to the calling thread and its action is to invoke a signal-catching function, or the  
 22737 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or  
 22738 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately  
 22739 and the calling process shall not be suspended.

22740 The suspension time caused by this function may be longer than requested because the  
 22741 argument value is rounded up to an integer multiple of the sleep resolution, or because of the  
 22742 scheduling of other activity by the system. But, except for the case of being interrupted by a  
 22743 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the  
 22744 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as  
 22745 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function  
 22746 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the  
 22747 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being  
 22748 interrupted by a signal.

22749 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any  
 22750 signal.

22751 The `clock_nanosleep()` function shall fail if the *clock\_id* argument refers to the CPU-time clock of  
 22752 the calling thread. It is unspecified whether *clock\_id* values of other CPU-time clocks are allowed.

22753 **RETURN VALUE**

22754 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value  
 22755 shall be zero.

22756 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall  
 22757 return the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp*  
 22758 argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the  
 22759 amount of time remaining in the interval (the requested time minus the time actually slept). If  
 22760 the *rmtp* argument is NULL, the remaining time is not returned. The absolute `clock_nanosleep()`  
 22761 function has no effect on the structure referenced by *rmtp*.

22762 If `clock_nanosleep()` fails, it shall return the corresponding error value.

**clock\_nanosleep()**22763 **ERRORS**22764 The *clock\_nanosleep()* function shall fail if:22765 [EINTR] The *clock\_nanosleep()* function was interrupted by a signal.

22766 [EINVAL] The *rqt*p argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in flags and the *rqt*p argument is outside the range for the clock specified by *clock\_id* or the *clock\_id* argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.

22771 [ENOTSUP] The *clock\_id* argument specifies a clock for which *clock\_nanosleep()* is not supported, such as a CPU-time clock.

22773 **EXAMPLES**

22774 None.

22775 **APPLICATION USAGE**

22776 Calling *clock\_nanosleep()* with the value `TIMER_ABSTIME` not set in the *flags* argument and with a *clock\_id* of `CLOCK_REALTIME` is equivalent to calling *nanosleep()* with the same *rqt*p and *rmt*p arguments.

22779 **RATIONALE**

22780 The *nanosleep()* function specifies that the system-wide clock `CLOCK_REALTIME` is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

22784 There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic process that is activated at time  $T_0$ , executes for a while, and then wants to suspend itself until time  $T_0+T$ , the period being  $T$ . If this process wants to use the *nanosleep()* function, it must first call *clock\_gettime()* to get the current time, then calculate the difference between the current time and  $T_0+T$  and, finally, call *nanosleep()* using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock\_nanosleep()* function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

22797 Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock\_nanosleep()* function in POSIX.1-2008 are the same as for the inclusion of the relative *nanosleep()*.

22801 It is also possible to implement precise periodic processes using *pthread\_cond\_timedwait()*, in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread\_cond\_timedwait()* is not available in implementations that do not support threads.

22807 Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock\_nanosleep()* function, the *rmt*p argument is only used in the relative sleep. This argument is needed in the relative *clock\_nanosleep()* function to reissue the function

22810 call if it is interrupted by a signal, but it is not needed in the absolute *clock\_nanosleep()* function  
22811 call; if the call is interrupted by a signal, the absolute *clock\_nanosleep()* function can be invoked  
22812 again with the same *rqt* argument used in the interrupted call.

22813 **FUTURE DIRECTIONS**

22814 None.

22815 **SEE ALSO**

22816 *clock\_getres()*, *nanosleep()*, *pthread\_cond\_timedwait()*, *sleep()*

22817 XBD <[time.h](#)>

22818 **CHANGE HISTORY**

22819 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

22820 **Issue 7**

22821 The *clock\_nanosleep()* function is moved from the Clock Selection option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**clock\_settime()**

System Interfaces

22822 **NAME**

22823 clock\_settime — clock and timer functions

22824 **SYNOPSIS**22825 CX `#include <time.h>`22826 `int clock_settime(clockid_t clock_id, const struct timespec *tp);`22827 **DESCRIPTION**22828 Refer to *clock\_getres()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

22829 **NAME**

22830 clog, clogf, clogl — complex natural logarithm functions

22831 **SYNOPSIS**

22832 #include &lt;complex.h&gt;

22833 double complex clog(double complex z);

22834 float complex clogf(float complex z);

22835 long double complex clogl(long double complex z);

22836 **DESCRIPTION**22837 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22838 conflict between the requirements described here and the ISO C standard is unintentional. This  
22839 volume of POSIX.1-2008 defers to the ISO C standard.22840 These functions shall compute the complex natural (base  $e$ ) logarithm of  $z$ , with a branch cut  
22841 along the negative real axis.22842 **RETURN VALUE**22843 These functions shall return the complex natural logarithm value, in the range of a strip  
22844 mathematically unbounded along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary  
22845 axis.22846 **ERRORS**

22847 No errors are defined.

22848 **EXAMPLES**

22849 None.

22850 **APPLICATION USAGE**

22851 None.

22852 **RATIONALE**

22853 None.

22854 **FUTURE DIRECTIONS**

22855 None.

22856 **SEE ALSO**22857 [cexp\(\)](#)22858 XBD [<complex.h>](#)22859 **CHANGE HISTORY**

22860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**close()**22861 **NAME**

22862 close — close a file descriptor

22863 **SYNOPSIS**

22864 #include &lt;unistd.h&gt;

22865 int close(int *fildev*);22866 **DESCRIPTION**

22867 The *close()* function shall deallocate the file descriptor indicated by *fildev*. To deallocate means to  
 22868 make the file descriptor available for return by subsequent calls to *open()* or other functions that  
 22869 allocate file descriptors. All outstanding record locks owned by the process on the file associated  
 22870 with the file descriptor shall be removed (that is, unlocked).

22871 If *close()* is interrupted by a signal that is to be caught, it shall return  $-1$  with *errno* set to [EINTR]  
 22872 and the state of *fildev* is unspecified. If an I/O error occurred while reading from or writing to  
 22873 the file system during *close()*, it may return  $-1$  with *errno* set to [EIO]; if this error is returned, the  
 22874 state of *fildev* is unspecified.

22875 When all file descriptors associated with a pipe or FIFO special file are closed, any data  
 22876 remaining in the pipe or FIFO shall be discarded.

22877 When all file descriptors associated with an open file description have been closed, the open file  
 22878 description shall be freed.

22879 If the link count of the file is 0, when all file descriptors associated with the file are closed, the  
 22880 space occupied by the file shall be freed and the file shall no longer be accessible.

22881 OB XSR If a STREAMS-based *fildev* is closed and the calling process was previously registered to receive  
 22882 a SIGPOLL signal for events associated with that STREAM, the calling process shall be  
 22883 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause  
 22884 the STREAM associated with *fildev* to be dismantled. If O\_NONBLOCK is not set and there have  
 22885 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*  
 22886 shall wait for an unspecified time (for each module and driver) for any output to drain before  
 22887 dismantling the STREAM. The time delay can be changed via an I\_SETCLTIME *ioctl()* request. If  
 22888 the O\_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for  
 22889 output to drain, and shall dismantle the STREAM immediately.

22890 If the implementation supports STREAMS-based pipes, and *fildev* is associated with one end of a  
 22891 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the  
 22892 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end  
 22893 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and  
 22894 gets detached, the STREAM associated with that end shall also be dismantled.

22895 XSI If *fildev* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal  
 22896 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is  
 22897 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal  
 22898 flushes all queued input and output.

22899 OB XSR If *fildev* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message  
 22900 may be sent to the master.

22901 When there is an outstanding cancelable asynchronous I/O operation against *fildev* when *close()*  
 22902 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes  
 22903 as if the *close()* operation had not yet occurred. All operations that are not canceled shall  
 22904 complete as if the *close()* blocked until the operations completed. The *close()* operation itself  
 22905 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and  
 22906 which I/O operation may be canceled upon *close()*, is implementation-defined.

22907 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,  
 22908 a process has it mapped), then the entire contents of the memory object shall persist until the  
 22909 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a  
 22910 shared memory object and the close results in the memory object becoming unreferenced, and  
 22911 the memory object has been unlinked, then the memory object shall be removed.

22912 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in  
 22913 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,  
 22914 and the socket has untransmitted data, then *close()* shall block for up to the current linger  
 22915 interval until all data is transmitted.

#### 22916 RETURN VALUE

22917 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 22918 indicate the error.

#### 22919 ERRORS

22920 The *close()* function shall fail if:

22921 [EBADF] The *fdes* argument is not a valid file descriptor.

22922 [EINTR] The *close()* function was interrupted by a signal.

22923 The *close()* function may fail if:

22924 [EIO] An I/O error occurred while reading from or writing to the file system.

#### 22925 EXAMPLES

##### 22926 Reassigning a File Descriptor

22927 The following example closes the file descriptor associated with standard output for the current  
 22928 process, re-assigns standard output to a new file descriptor, and closes the original file descriptor  
 22929 to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard  
 22930 input) is not closed.

```
22931 #include <unistd.h>
22932 ...
22933 int pfd;
22934 ...
22935 close(1);
22936 dup(pfd);
22937 close(pfd);
22938 ...
```

22939 Incidentally, this is exactly what could be achieved using:

```
22940 dup2(pfd, 1);
22941 close(pfd);
```

**close()**22942 **Closing a File Descriptor**

22943 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is  
 22944 made to associate that file descriptor with a stream.

```

22945 #include <stdio.h>
22946 #include <unistd.h>
22947 #include <stdlib.h>

22948 #define LOCKFILE "/etc/ptmp"
22949 ...
22950 int pfd;
22951 FILE *fpfd;
22952 ...
22953 if ((fpfd = fdopen (pfd, "w")) == NULL) {
22954     close(pfd);
22955     unlink(LOCKFILE);
22956     exit(1);
22957 }
22958 ...
  
```

22959 **APPLICATION USAGE**

22960 An application that had used the *stdio* routine *fopen()* to open a file should use the  
 22961 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no  
 22962 longer exists, since the integer corresponding to it no longer refers to a file.

22963 **RATIONALE**

22964 The use of interruptible device close routines should be discouraged to avoid problems with the  
 22965 implicit closes of file descriptors by *exec* and *exit()*. This volume of POSIX.1-2008 only intends to  
 22966 permit such behavior by specifying the [EINTR] error condition.

22967 Note that the requirement for *close()* on a socket to block for up to the current linger interval is  
 22968 not conditional on the O\_NONBLOCK setting.

22969 **FUTURE DIRECTIONS**

22970 None.

22971 **SEE ALSO**

22972 [Section 2.6](#) (on page 494), *exec*, *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, *unlink()*

22973 XBD [<unistd.h>](#)

22974 **CHANGE HISTORY**

22975 First released in Issue 1. Derived from Issue 1 of the SVID.

22976 **Issue 5**

22977 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

22978 **Issue 6**

22979 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of  
 22980 the XSI STREAMS Option Group.

22981 The following new requirements on POSIX implementations derive from alignment with the  
 22982 Single UNIX Specification:

- 22983 • The [EIO] error condition is added as an optional error.

- 22984                   • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as  
22985                   unspecified if an I/O error occurs and an [EIO] error condition is returned.
- 22986                   Text referring to sockets is added to the DESCRIPTION.
- 22987                   The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
22988                   shared memory objects and memory mapped files (and not typed memory objects) are the types  
22989                   of memory objects to which the paragraph on last closes applies.
- 22990                   IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded  
22991                   text relating to the master side of a pseudo-terminal. The reason for the change is that the  
22992                   behavior of pseudo-terminals and regular terminals should be as much alike as possible in this  
22993                   case; the change achieves that and matches historical behavior.
- 22994   **Issue 7**
- 22995                   Functionality relating to the XSI STREAMS option is marked obsolescent.
- 22996                   Functionality relating to the Asynchronous Input and Output and Memory Mapped Files  
22997                   options is moved to the Base.
- 22998                   Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for  
22999                   *close()* on a socket to block for up to the current linger interval is not conditional on the  
23000                   O\_NONBLOCK setting.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**closedir()**23001 **NAME**

23002           closedir — close a directory stream

23003 **SYNOPSIS**

```
23004       #include <dirent.h>
23005       int closedir(DIR *dirp);
```

23006 **DESCRIPTION**

23007       The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon  
 23008       return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file  
 23009       descriptor is used to implement type **DIR**, that file descriptor shall be closed.

23010 **RETURN VALUE**

23011       Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*  
 23012       set to indicate the error.

23013 **ERRORS**

23014       The *closedir()* function may fail if:

- |       |         |                                                                      |
|-------|---------|----------------------------------------------------------------------|
| 23015 | [EBADF] | The <i>dirp</i> argument does not refer to an open directory stream. |
| 23016 | [EINTR] | The <i>closedir()</i> function was interrupted by a signal.          |

23017 **EXAMPLES**23018           **Closing a Directory Stream**

23019       The following program fragment demonstrates how the *closedir()* function is used.

```
23020       ...
23021       DIR *dir;
23022       struct dirent *dp;
23023       ...
23024       if ((dir = opendir (".")) == NULL) {
23025       ...
23026       }
23027       while ((dp = readdir (dir)) != NULL) {
23028       ...
23029       }
23030       closedir(dir);
23031       ...

```

23032 **APPLICATION USAGE**

23033       None.

23034 **RATIONALE**

23035       None.

23036 **FUTURE DIRECTIONS**

23037       None.

23038 **SEE ALSO**

23039       *dirfd()*, *fdopendir()*

23040       XBD <dirent.h>

23041 **CHANGE HISTORY**

23042 First released in Issue 2.

23043 **Issue 6**23044 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.23045 The following new requirements on POSIX implementations derive from alignment with the  
23046 Single UNIX Specification:

- 23047 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
23048 required for conforming implementations of previous POSIX specifications, it was not  
23049 required for UNIX applications.
- 23050 • The [EINTR] error condition is added as an optional error condition.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**closelog()**23051 **NAME**

23052 closelog, openlog, setlogmask, syslog — control system log

23053 **SYNOPSIS**

```

23054 XSI #include <syslog.h>
23055 void closelog(void);
23056 void openlog(const char *ident, int logopt, int facility);
23057 int setlogmask(int maskpri);
23058 void syslog(int priority, const char *message, ... /* arguments */);

```

23059 **DESCRIPTION**

23060 The *syslog()* function shall send a message to an implementation-defined logging facility, which  
 23061 may log it in an implementation-defined system log, write it to the system console, forward it to  
 23062 a list of users, or forward it to the logging facility on another host over the network. The logged  
 23063 message shall include a message header and a message body. The message header contains at  
 23064 least a timestamp and a tag string.

23065 The message body is generated from the *message* and following arguments in the same manner  
 23066 as if these were arguments to *printf()*, except that the additional conversion specification `%m`  
 23067 shall be recognized; it shall convert no arguments, shall cause the output of the error message  
 23068 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument  
 23069 specifications of the "`%n$`" form. If a complete conversion specification with the *m* conversion  
 23070 specifier character is not just `%m`, the behavior is undefined. A trailing `<newline>` may be added  
 23071 if needed.

23072 Values of the *priority* argument are formed by OR'ing together a severity-level value and an  
 23073 optional facility value. If no facility value is specified, the current default facility value is used.

23074 Possible values of severity level include:

|       |             |                                                                                                 |
|-------|-------------|-------------------------------------------------------------------------------------------------|
| 23075 | LOG_EMERG   | A panic condition.                                                                              |
| 23076 | LOG_ALERT   | A condition that should be corrected immediately, such as a corrupted system<br>23077 database. |
| 23078 | LOG_CRIT    | Critical conditions, such as hard device errors.                                                |
| 23079 | LOG_ERR     | Errors.                                                                                         |
| 23080 | LOG_WARNING | Warning messages.                                                                               |
| 23081 |             |                                                                                                 |
| 23082 | LOG_NOTICE  | Conditions that are not error conditions, but that may require special<br>23083 handling.       |
| 23084 | LOG_INFO    | Informational messages.                                                                         |
| 23085 | LOG_DEBUG   | Messages that contain information normally of use only when debugging a<br>23086 program.       |

23087 The facility indicates the application or system component generating the message. Possible  
 23088 facility values include:

|       |            |                                                                                                                   |
|-------|------------|-------------------------------------------------------------------------------------------------------------------|
| 23089 | LOG_USER   | Messages generated by arbitrary processes. This is the default facility<br>23090 identifier if none is specified. |
| 23091 | LOG_LOCAL0 | Reserved for local use.                                                                                           |

|       |                     |                                                                                                                                  |
|-------|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| 23092 | LOG_LOCAL1          | Reserved for local use.                                                                                                          |
| 23093 | LOG_LOCAL2          | Reserved for local use.                                                                                                          |
| 23094 | LOG_LOCAL3          | Reserved for local use.                                                                                                          |
| 23095 | LOG_LOCAL4          | Reserved for local use.                                                                                                          |
| 23096 | LOG_LOCAL5          | Reserved for local use.                                                                                                          |
| 23097 | LOG_LOCAL6          | Reserved for local use.                                                                                                          |
| 23098 | LOG_LOCAL7          | Reserved for local use.                                                                                                          |
| 23099 |                     | The <i>openlog()</i> function shall set process attributes that affect subsequent calls to <i>syslog()</i> . The                 |
| 23100 |                     | <i>ident</i> argument is a string that is prepended to every message. The <i>logopt</i> argument indicates                       |
| 23101 |                     | logging options. Values for <i>logopt</i> are constructed by a bitwise-inclusive OR of zero or more of                           |
| 23102 |                     | the following:                                                                                                                   |
| 23103 | LOG_PID             | Log the process ID with each message. This is useful for identifying specific                                                    |
| 23104 |                     | processes.                                                                                                                       |
| 23105 | LOG_CONS            | Write messages to the system console if they cannot be sent to the logging                                                       |
| 23106 |                     | facility. The <i>syslog()</i> function ensures that the process does not acquire the                                             |
| 23107 |                     | console as a controlling terminal in the process of writing the message.                                                         |
| 23108 | LOG_NDELAY          | Open the connection to the logging facility immediately. Normally the open is                                                    |
| 23109 |                     | delayed until the first message is logged. This is useful for programs that need                                                 |
| 23110 |                     | to manage the order in which file descriptors are allocated.                                                                     |
| 23111 | LOG_ODELAY          | Delay open until <i>syslog()</i> is called.                                                                                      |
| 23112 | LOG_NOWAIT          | Do not wait for child processes that may have been created during the course                                                     |
| 23113 |                     | of logging the message. This option should be used by processes that enable                                                      |
| 23114 |                     | notification of child termination using SIGCHLD, since <i>syslog()</i> may                                                       |
| 23115 |                     | otherwise block waiting for a child whose exit status has already been                                                           |
| 23116 |                     | collected.                                                                                                                       |
| 23117 |                     | The <i>facility</i> argument encodes a default facility to be assigned to all messages that do not have an                       |
| 23118 |                     | explicit facility already encoded. The initial default facility is LOG_USER.                                                     |
| 23119 |                     | The <i>openlog()</i> and <i>syslog()</i> functions may allocate a file descriptor. It is not necessary to call                   |
| 23120 |                     | <i>openlog()</i> prior to calling <i>syslog()</i> .                                                                              |
| 23121 |                     | The <i>closelog()</i> function shall close any open file descriptors allocated by previous calls to                              |
| 23122 |                     | <i>openlog()</i> or <i>syslog()</i> .                                                                                            |
| 23123 |                     | The <i>setlogmask()</i> function shall set the log priority mask for the current process to <i>maskpri</i> and                   |
| 23124 |                     | return the previous mask. If the <i>maskpri</i> argument is 0, the current log mask is not modified.                             |
| 23125 |                     | Calls by the current process to <i>syslog()</i> with a priority not set in <i>maskpri</i> shall be rejected. The                 |
| 23126 |                     | default log mask allows all priorities to be logged. A call to <i>openlog()</i> is not required prior to                         |
| 23127 |                     | calling <i>setlogmask()</i> .                                                                                                    |
| 23128 |                     | Symbolic constants for use as values of the <i>logopt</i> , <i>facility</i> , <i>priority</i> , and <i>maskpri</i> arguments are |
| 23129 |                     | defined in the <b>&lt;syslog.h&gt;</b> header.                                                                                   |
| 23130 | <b>RETURN VALUE</b> |                                                                                                                                  |
| 23131 |                     | The <i>setlogmask()</i> function shall return the previous log priority mask. The <i>closelog()</i> , <i>openlog()</i> ,         |
| 23132 |                     | and <i>syslog()</i> functions shall not return a value.                                                                          |

**closelog()**23133 **ERRORS**

23134           None.

23135 **EXAMPLES**23136           **Using openlog()**

23137           The following example causes subsequent calls to *syslog()* to log the process ID with each  
 23138           message, and to write messages to the system console if they cannot be sent to the logging  
 23139           facility.

```
23140           #include <syslog.h>

23141           char *ident = "Process demo";
23142           int logopt = LOG_PID | LOG_CONS;
23143           int facility = LOG_USER;
23144           ...
23145           openlog(ident, logopt, facility);
```

23146           **Using setlogmask()**

23147           The following example causes subsequent calls to *syslog()* to accept error messages, and to reject  
 23148           all other messages.

```
23149           #include <syslog.h>

23150           int result;
23151           int mask = LOG_MASK (LOG_ERR);
23152           ...
23153           result = setlogmask(mask);
```

23154           **Using syslog**

23155           The following example sends the message "This is a message" to the default logging  
 23156           facility, marking the message as an error message generated by random processes.

```
23157           #include <syslog.h>

23158           char *message = "This is a message";
23159           int priority = LOG_ERR | LOG_USER;
23160           ...
23161           syslog(priority, message);
```

23162 **APPLICATION USAGE**

23163           None.

23164 **RATIONALE**

23165           None.

23166 **FUTURE DIRECTIONS**

23167           None.

23168 **SEE ALSO**23169           *fprintf()*23170           XBD *<syslog.h>*

23171 **CHANGE HISTORY**

23172 First released in Issue 4, Version 2.

23173 **Issue 5**

23174 Moved from X/OPEN UNIX extension to BASE.

23175 **Issue 6**23176 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES  
23177 section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**confstr()**23178 **NAME**23179 `confstr` — get configurable variables23180 **SYNOPSIS**23181 `#include <unistd.h>`23182 `size_t confstr(int name, char *buf, size_t len);`23183 **DESCRIPTION**23184 The `confstr()` function shall return configuration-defined string values. Its use and purpose are  
23185 similar to `sysconf()`, but it is used where string values rather than numeric values are returned.23186 The `name` argument represents the system variable to be queried. The implementation shall  
23187 support the following name values, defined in `<unistd.h>`. It may support others:

23188 `_CS_PATH`  
 23189 `_CS_POSIX_V7_ILP32_OFF32_CFLAGS`  
 23190 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`  
 23191 `_CS_POSIX_V7_ILP32_OFF32_LIBS`  
 23192 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`  
 23193 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`  
 23194 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`  
 23195 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`  
 23196 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`  
 23197 `_CS_POSIX_V7_LP64_OFF64_LIBS`  
 23198 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`  
 23199 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`  
 23200 `_CS_POSIX_V7_LPBIG_OFFBIG_LIBS`  
 23201 `_CS_POSIX_V7_THREADS_CFLAGS`  
 23202 `_CS_POSIX_V7_THREADS_LDFLAGS`  
 23203 `_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS`  
 23204 `_CS_V7_ENV`

23205 **OR** `_CS_POSIX_V6_ILP32_OFF32_CFLAGS`  
 23206 `_CS_POSIX_V6_ILP32_OFF32_LDFLAGS`  
 23207 `_CS_POSIX_V6_ILP32_OFF32_LIBS`  
 23208 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS`  
 23209 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS`  
 23210 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS`  
 23211 `_CS_POSIX_V6_LP64_OFF64_CFLAGS`  
 23212 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`  
 23213 `_CS_POSIX_V6_LP64_OFF64_LIBS`  
 23214 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS`  
 23215 `_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS`  
 23216 `_CS_POSIX_V6_LPBIG_OFFBIG_LIBS`  
 23217 `_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS`  
 23218 `_CS_V6_ENV`

23219 If `len` is not 0, and if `name` has a configuration-defined value, `confstr()` shall copy that value into  
 23220 the `len`-byte buffer pointed to by `buf`. If the string to be returned is longer than `len` bytes,  
 23221 including the terminating null, then `confstr()` shall truncate the string to `len-1` bytes and null-  
 23222 terminate the result. The application can detect that the string was truncated by comparing the  
 23223 value returned by `confstr()` with `len`.

23224 If `len` is 0 and `buf` is a null pointer, then `confstr()` shall still return the integer value as defined  
 23225 below, but shall not return a string. If `len` is 0 but `buf` is not a null pointer, the result is

23226 unspecified.

23227 After a call to:

23228 `confstr(_CS_V7_ENV, buf, sizeof(buf))`

23229 the string stored in *buf* will contain the <space>-separated list of variable=value environment  
23230 variable pairs required by the implementation to create a conforming environment, as described  
23231 in the implementations' conformance documentation.

23232 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

23233 `confstr(_CS_PATH, buf, sizeof(buf))`

23234 can be used as a value of the *PATH* environment variable that accesses all of the standard  
23235 utilities of POSIX.1-2008, if the return value is less than or equal to *sizeof(buf)*.

#### 23236 RETURN VALUE

23237 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be  
23238 needed to hold the entire configuration-defined value including the terminating null. If this  
23239 return value is greater than *len*, the string returned in *buf* is truncated.

23240 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

23241 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*  
23242 unchanged.

#### 23243 ERRORS

23244 The *confstr()* function shall fail if:

23245 [EINVAL] The value of the *name* argument is invalid.

#### 23246 EXAMPLES

23247 None.

#### 23248 APPLICATION USAGE

23249 An application can distinguish between an invalid *name* parameter value and one that  
23250 corresponds to a configurable variable that has no configuration-defined value by checking if  
23251 *errno* is modified. This mirrors the behavior of *sysconf()*.

23252 The original need for this function was to provide a way of finding the configuration-defined  
23253 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to  
23254 include directories that could contain utilities replacing the standard utilities in the Shell and  
23255 Utilities volume of POSIX.1-2008, applications need a way to determine the system-supplied  
23256 *PATH* environment variable value that contains the correct search path for the standard utilities.

23257 An application could use:

23258 `confstr(name, (char *)NULL, (size_t)0)`

23259 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to  
23260 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,  
23261 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use  
23262 *malloc()* to allocate a larger buffer if it finds that this is too small.

#### 23263 RATIONALE

23264 Application developers can normally determine any configuration variable by means of reading  
23265 from the stream opened by a call to:

23266 `popen("command -p getconf variable", "r");`

23267 The *confstr()* function with a *name* argument of *\_CS\_PATH* returns a string that can be used as a

**confstr()**

- 23268 *PATH* environment variable setting that will reference the standard shell and utilities as  
23269 described in the Shell and Utilities volume of POSIX.1-2008.
- 23270 The *confstr()* function copies the returned string into a buffer supplied by the application instead  
23271 of returning a pointer to a string. This allows a cleaner function in some implementations (such  
23272 as those with lightweight threads) and resolves questions about when the application must copy  
23273 the string returned.
- 23274 **FUTURE DIRECTIONS**
- 23275 None.
- 23276 **SEE ALSO**
- 23277 *exec*, *fpathconf()*, *sysconf()*
- 23278 XBD <*unistd.h*>
- 23279 XCU *c99*
- 23280 **CHANGE HISTORY**
- 23281 First released in Issue 4. Derived from the ISO POSIX-2 standard.
- 23282 **Issue 5**
- 23283 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those  
23284 marked EX are new in this version.
- 23285 **Issue 6**
- 23286 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the  
23287 size of the buffer now explicitly states that this includes the terminating null.
- 23288 The following new requirements on POSIX implementations derive from alignment with the  
23289 Single UNIX Specification:
- 23290 • The DESCRIPTION is updated with new arguments which can be used to determine
  - 23291 configuration strings for C compiler flags, linker/loader flags, and libraries for each
  - 23292 different supported programming environment. This is a change to support data size
  - 23293 neutrality.
- 23294 The following changes were made to align with the IEEE P1003.1a draft standard:
- 23295 • The DESCRIPTION is updated to include text describing how *\_CS\_PATH* can be used to
  - 23296 obtain a *PATH* to access the standard utilities.
- 23297 The macros associated with the *c89* programming models are marked LEGACY and new  
23298 equivalent macros associated with *c99* are introduced.
- 23299 **Issue 7**
- 23300 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the *\_CS\_V7\_ENV* variable.
- 23301 Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag  
23302 to enable threads.
- 23303 The V6 variables for the supported programming environments are marked obsolescent.
- 23304 The variables for the supported programming environments are updated to be V7.
- 23305 The LEGACY variables and obsolescent values are removed.

23306 **NAME**

23307 conj, conjf, conjl — complex conjugate functions

23308 **SYNOPSIS**

23309 #include &lt;complex.h&gt;

23310 double complex conj(double complex z);

23311 float complex conjf(float complex z);

23312 long double complex conjl(long double complex z);

23313 **DESCRIPTION**

23314 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23315 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23316 volume of POSIX.1-2008 defers to the ISO C standard.

23317 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary  
 23318 part.

23319 **RETURN VALUE**

23320 These functions return the complex conjugate value.

23321 **ERRORS**

23322 No errors are defined.

23323 **EXAMPLES**

23324 None.

23325 **APPLICATION USAGE**

23326 None.

23327 **RATIONALE**

23328 None.

23329 **FUTURE DIRECTIONS**

23330 None.

23331 **SEE ALSO**23332 [carg\(\)](#), [cimag\(\)](#), [cproj\(\)](#), [creal\(\)](#)23333 XBD [<complex.h>](#)23334 **CHANGE HISTORY**

23335 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**connect()**

System Interfaces

23336 **NAME**

23337 connect — connect a socket

23338 **SYNOPSIS**

23339 #include &lt;sys/socket.h&gt;

23340 int connect(int *socket*, const struct sockaddr \**address*,  
23341 socklen\_t *address\_len*);23342 **DESCRIPTION**23343 The *connect()* function shall attempt to make a connection on a connection-mode socket or to set  
23344 or reset the peer address of a connectionless-mode socket. The function takes the following  
23345 arguments:

|       |                    |                                                                                                                                                     |
|-------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 23346 | <i>socket</i>      | Specifies the file descriptor associated with the socket.                                                                                           |
| 23347 | <i>address</i>     | Points to a <b>sockaddr</b> structure containing the peer address. The length and format of the address depend on the address family of the socket. |
| 23349 | <i>address_len</i> | Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i> argument.                                                    |

23351 If the socket has not already been bound to a local address, *connect()* shall bind it to an address  
23352 which, unless the socket's address family is AF\_UNIX, is an unused local address.23353 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,  
23354 and no connection is made. For SOCK\_DGRAM sockets, the peer address identifies where all  
23355 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent  
23356 *recv()* functions. If the *sa\_family* member of *address* is AF\_UNSPEC, the socket's peer address  
23357 shall be reset. Note that despite no connection being made, the term "connected" is used to  
23358 describe a connectionless-mode socket for which a peer address has been set.23359 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection  
23360 to the address specified by the *address* argument. If the connection cannot be established  
23361 immediately and O\_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall  
23362 block for up to an unspecified timeout interval until the connection is established. If the timeout  
23363 interval expires before the connection is established, *connect()* shall fail and the connection  
23364 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked  
23365 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection  
23366 request shall not be aborted, and the connection shall be established asynchronously.23367 If the connection cannot be established immediately and O\_NONBLOCK is set for the file  
23368 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection  
23369 request shall not be aborted, and the connection shall be established asynchronously. Subsequent  
23370 calls to *connect()* for the same socket, before the connection is established, shall fail and set *errno*  
23371 to [EALREADY].23372 When the connection has been established asynchronously, *pselect()*, *select()*, and *poll()* shall  
23373 indicate that the file descriptor for the socket is ready for writing.23374 The socket in use may require the process to have appropriate privileges to use the *connect()*  
23375 function.23376 **RETURN VALUE**23377 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*  
23378 set to indicate the error.

23379 **ERRORS**

- 23380 The *connect()* function shall fail if:
- 23381 [EADDRNOTAVAIL]  
23382 The specified address is not available from the local machine.
- 23383 [EAFNOSUPPORT]  
23384 The specified address is not a valid address for the address family of the  
23385 specified socket.
- 23386 [EALREADY] A connection request is already in progress for the specified socket.
- 23387 [EBADF] The *socket* argument is not a valid file descriptor.
- 23388 [ECONNREFUSED]  
23389 The target address was not listening for connections or refused the connection  
23390 request.
- 23391 [EINPROGRESS] O\_NONBLOCK is set for the file descriptor for the socket and the connection  
23392 cannot be immediately established; the connection shall be established  
23393 asynchronously.
- 23394 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal  
23395 that was caught; the connection shall be established asynchronously.
- 23396 [EISCONN] The specified socket is connection-mode and is already connected.
- 23397 [ENETUNREACH]  
23398 No route to the network is present.
- 23399 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 23400 [EPROTOTYPE] The specified address has a different type than the socket bound to the  
23401 specified peer address.
- 23402 [ETIMEDOUT] The attempt to connect timed out before a connection was made.
- 23403 If the address family of the socket is AF\_UNIX, then *connect()* shall fail if:
- 23404 [EIO] An I/O error occurred while reading from or writing to the file system.
- 23405 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
23406 in *address*.
- 23407 [ENAMETOOLONG]  
23408 The length of a component of a pathname is longer than {NAME\_MAX}.
- 23409 [ENOENT] A component of the pathname does not name an existing file or the pathname  
23410 is an empty string.
- 23411 [ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory, or  
23412 the pathname in *address* contains at least one non-*<slash>* character and ends  
23413 with one or more trailing *<slash>* characters and the last pathname  
23414 component names an existing file that is neither a directory nor a symbolic  
23415 link to a directory.
- 23416 The *connect()* function may fail if:
- 23417 [EACCES] Search permission is denied for a component of the path prefix; or write access  
23418 to the named socket is denied.

**connect()**

System Interfaces

|       |                          |                                                                                                                                                                                     |
|-------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 23419 | [EADDRINUSE]             | Attempt to establish a connection that uses addresses that are already in use.                                                                                                      |
| 23420 | [ECONNRESET]             | Remote host reset the connection request.                                                                                                                                           |
| 23421 | [EHOSTUNREACH]           |                                                                                                                                                                                     |
| 23422 |                          | The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).                                                                      |
| 23423 |                          |                                                                                                                                                                                     |
| 23424 | [EINVAL]                 | The <i>address_len</i> argument is not a valid length for the address family, or invalid address family in the <b>sockaddr</b> structure.                                           |
| 23425 |                          |                                                                                                                                                                                     |
| 23426 | [ELOOP]                  | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .                                                                       |
| 23427 |                          |                                                                                                                                                                                     |
| 23428 | [ENAMETOOLONG]           |                                                                                                                                                                                     |
| 23429 |                          | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.                       |
| 23430 |                          |                                                                                                                                                                                     |
| 23431 |                          |                                                                                                                                                                                     |
| 23432 | [ENETDOWN]               | The local network interface used to reach the destination is down.                                                                                                                  |
| 23433 | [ENOBUFS]                | No buffer space is available.                                                                                                                                                       |
| 23434 | [EOPNOTSUPP]             | The socket is listening and cannot be connected.                                                                                                                                    |
| 23435 | <b>EXAMPLES</b>          |                                                                                                                                                                                     |
| 23436 |                          | None.                                                                                                                                                                               |
| 23437 | <b>APPLICATION USAGE</b> |                                                                                                                                                                                     |
| 23438 |                          | If <i>connect()</i> fails, the state of the socket is unspecified. Conforming applications should close the file descriptor and create a new socket before attempting to reconnect. |
| 23439 |                          |                                                                                                                                                                                     |
| 23440 | <b>RATIONALE</b>         |                                                                                                                                                                                     |
| 23441 |                          | None.                                                                                                                                                                               |
| 23442 | <b>FUTURE DIRECTIONS</b> |                                                                                                                                                                                     |
| 23443 |                          | None.                                                                                                                                                                               |
| 23444 | <b>SEE ALSO</b>          |                                                                                                                                                                                     |
| 23445 |                          | <i>accept()</i> , <i>bind()</i> , <i>close()</i> , <i>getsockname()</i> , <i>poll()</i> , <i>pselect()</i> , <i>send()</i> , <i>shutdown()</i> , <i>socket()</i>                    |
| 23446 |                          | XBD <sys/socket.h>                                                                                                                                                                  |
| 23447 | <b>CHANGE HISTORY</b>    |                                                                                                                                                                                     |
| 23448 |                          | First released in Issue 6. Derived from the XNS, Issue 5.2 specification.                                                                                                           |
| 23449 |                          | The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.                                                            |
| 23450 |                          |                                                                                                                                                                                     |
| 23451 | <b>Issue 7</b>           |                                                                                                                                                                                     |
| 23452 |                          | Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.                                                                           |
| 23453 |                          |                                                                                                                                                                                     |
| 23454 |                          | Austin Group Interpretation 1003.1-2001 #143 is applied.                                                                                                                            |
| 23455 |                          | Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a peer address for a datagram socket.                                                    |
| 23456 |                          |                                                                                                                                                                                     |
| 23457 |                          | SD5-XSH-ERN-185 is applied.                                                                                                                                                         |

23458 **NAME**

23459 copysign, copysignf, copysignl — number manipulation function

23460 **SYNOPSIS**

23461 #include &lt;math.h&gt;

23462 double copysign(double *x*, double *y*);23463 float copysignf(float *x*, float *y*);23464 long double copysignl(long double *x*, long double *y*);23465 **DESCRIPTION**

23466 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23467 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23468 volume of POSIX.1-2008 defers to the ISO C standard.

23469 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On  
 23470 implementations that represent a signed zero but do not treat negative zero consistently in  
 23471 arithmetic operations, these functions regard the sign of zero as positive.

23472 **RETURN VALUE**

23473 Upon successful completion, these functions shall return a value with the magnitude of *x* and  
 23474 the sign of *y*.

23475 **ERRORS**

23476 No errors are defined.

23477 **EXAMPLES**

23478 None.

23479 **APPLICATION USAGE**

23480 None.

23481 **RATIONALE**

23482 None.

23483 **FUTURE DIRECTIONS**

23484 None.

23485 **SEE ALSO**23486 [signbit\(\)](#)23487 XBD [<math.h>](#)23488 **CHANGE HISTORY**

23489 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**cos()**23490 **NAME**

23491 cos, cosf, cosl — cosine function

23492 **SYNOPSIS**

```
23493 #include <math.h>
23494 double cos(double x);
23495 float cosf(float x);
23496 long double cosl(long double x);
```

23497 **DESCRIPTION**

23498 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23499 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23500 volume of POSIX.1-2008 defers to the ISO C standard.

23501 These functions shall compute the cosine of their argument  $x$ , measured in radians.

23502 An application wishing to check for error situations should set *errno* to zero and call  
 23503 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23504 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23505 zero, an error has occurred.

23506 **RETURN VALUE**

23507 Upon successful completion, these functions shall return the cosine of  $x$ .

23508 **MX** If  $x$  is NaN, a NaN shall be returned.

23509 If  $x$  is  $\pm 0$ , the value 1.0 shall be returned.

23510 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 23511 defined value shall be returned.

23512 **ERRORS**

23513 These functions shall fail if:

23514 **MX** **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ .

23515 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 23516 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 23517 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23518 shall be raised.

23519 **EXAMPLES**23520 **Taking the Cosine of a 45-Degree Angle**

```
23521 #include <math.h>
23522 .
23523 double radians = 45 * M_PI / 180;
23524 double result;
23525 ...
23526 result = cos(radians);
```

23527 **APPLICATION USAGE**

23528 These functions may lose accuracy when their argument is near an odd multiple of  $\pi/2$  or is far  
 23529 from 0.

23530 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 23531 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23532 **RATIONALE**

23533 None.

23534 **FUTURE DIRECTIONS**

23535 None.

23536 **SEE ALSO**23537 *acos()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *sin()*, *tan()*23538 XBD Section 4.19 (on page 116), **<math.h>**23539 **CHANGE HISTORY**

23540 First released in Issue 1. Derived from Issue 1 of the SVID.

23541 **Issue 5**23542 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23543 text was previously published in the APPLICATION USAGE section.23544 **Issue 6**23545 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.23546 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23547 revised to align with the ISO/IEC 9899:1999 standard.23548 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23549 marked.

**cosh()**23550 **NAME**

23551 cosh, coshf, coshl — hyperbolic cosine functions

23552 **SYNOPSIS**

```
23553 #include <math.h>
23554 double cosh(double x);
23555 float coshf(float x);
23556 long double coshl(long double x);
```

23557 **DESCRIPTION**

23558 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23559 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23560 volume of POSIX.1-2008 defers to the ISO C standard.

23561 These functions shall compute the hyperbolic cosine of their argument *x*.

23562 An application wishing to check for error situations should set *errno* to zero and call  
 23563 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23564 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23565 zero, an error has occurred.

23566 **RETURN VALUE**23567 Upon successful completion, these functions shall return the hyperbolic cosine of *x*.

23568 If the correct value would cause overflow, a range error shall occur and *cosh()*, *coshf()*, and  
 23569 *coshl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 23570 respectively.

23571 **MX** If *x* is NaN, a NaN shall be returned.23572 If *x* is  $\pm 0$ , the value 1.0 shall be returned.23573 If *x* is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.23574 **ERRORS**

23575 These functions shall fail if:

23576 **Range Error** The result would cause an overflow.

23577 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 23578 then *errno* shall be set to [ERANGE]. If the integer expression  
 23579 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 23580 floating-point exception shall be raised.

23581 **EXAMPLES**

23582 None.

23583 **APPLICATION USAGE**

23584 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 23585 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23586 For IEEE Std 754-1985 **double**,  $710.5 < |x|$  implies that *cosh(x)* has overflowed.23587 **RATIONALE**

23588 None.

23589 **FUTURE DIRECTIONS**

23590 None.

23591 **SEE ALSO**23592 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*23593 XBD Section 4.19 (on page 116), **<math.h>**23594 **CHANGE HISTORY**

23595 First released in Issue 1. Derived from Issue 1 of the SVID.

23596 **Issue 5**23597 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23598 text was previously published in the APPLICATION USAGE section.23599 **Issue 6**23600 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.23601 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23602 revised to align with the ISO/IEC 9899:1999 standard.23603 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23604 marked.

**cosl()***System Interfaces*23605 **NAME**

23606           cosl — cosine function

23607 **SYNOPSIS**

23608           #include &lt;math.h&gt;

23609           long double cosl(long double x);

23610 **DESCRIPTION**23611           Refer to *cos()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

23612 **NAME**

23613 cpow, cpowf, cpowl — complex power functions

23614 **SYNOPSIS**

```
23615 #include <complex.h>
23616 double complex cpow(double complex x, double complex y);
23617 float complex cpowf(float complex x, float complex y);
23618 long double complex cpowl(long double complex x,
23619 long double complex y);
```

23620 **DESCRIPTION**

23621 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23622 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23623 volume of POSIX.1-2008 defers to the ISO C standard.

23624 These functions shall compute the complex power function  $x^y$ , with a branch cut for the first  
 23625 parameter along the negative real axis.

23626 **RETURN VALUE**

23627 These functions shall return the complex power function value.

23628 **ERRORS**

23629 No errors are defined.

23630 **EXAMPLES**

23631 None.

23632 **APPLICATION USAGE**

23633 None.

23634 **RATIONALE**

23635 None.

23636 **FUTURE DIRECTIONS**

23637 None.

23638 **SEE ALSO**23639 [cabs\(\)](#), [csqrt\(\)](#)23640 XBD [<complex.h>](#)23641 **CHANGE HISTORY**

23642 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**cproj()**23643 **NAME**23644 `cproj`, `cprojf`, `cprojl` — complex projection functions23645 **SYNOPSIS**

```
23646 #include <complex.h>
23647 double complex cproj(double complex z);
23648 float complex cprojf(float complex z);
23649 long double complex cprojl(long double complex z);
```

23650 **DESCRIPTION**

23651 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23652 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23653 volume of POSIX.1-2008 defers to the ISO C standard.

23654 These functions shall compute a projection of  $z$  onto the Riemann sphere:  $z$  projects to  $z$ , except  
 23655 that all complex infinities (even those with one infinite part and one NaN part) project to  
 23656 positive infinity on the real axis. If  $z$  has an infinite part, then  $cproj(z)$  shall be equivalent to:

```
23657 INFINITY + I * copysign(0.0, cimag(z))
```

23658 **RETURN VALUE**

23659 These functions shall return the value of the projection onto the Riemann sphere.

23660 **ERRORS**

23661 No errors are defined.

23662 **EXAMPLES**

23663 None.

23664 **APPLICATION USAGE**

23665 None.

23666 **RATIONALE**

23667 Two topologies are commonly used in complex mathematics: the complex plane with its  
 23668 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is  
 23669 better suited for transcendental functions, the Riemann sphere for algebraic functions. The  
 23670 complex types with their multiplicity of infinities provide a useful (though imperfect) model for  
 23671 the complex plane. The `cproj()` function helps model the Riemann sphere by mapping all  
 23672 infinities to one, and should be used just before any operation, especially comparisons, that  
 23673 might give spurious results for any of the other infinities. Note that a complex value with one  
 23674 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is  
 23675 infinite, the complex value is infinite independent of the value of the other part. For the same  
 23676 reason, `cabs()` returns an infinity if its argument has an infinite part and a NaN part.

23677 **FUTURE DIRECTIONS**

23678 None.

23679 **SEE ALSO**23680 `carg()`, `cimag()`, `conj()`, `creal()`23681 XBD `<complex.h>`23682 **CHANGE HISTORY**

23683 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23684 **NAME**

23685 creal, crealf, creall — complex real functions

23686 **SYNOPSIS**

23687 #include &lt;complex.h&gt;

23688 double creal(double complex z);

23689 float crealf(float complex z);

23690 long double creall(long double complex z);

23691 **DESCRIPTION**

23692 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23693 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23694 volume of POSIX.1-2008 defers to the ISO C standard.

23695 These functions shall compute the real part of z.

23696 **RETURN VALUE**

23697 These functions shall return the real part value.

23698 **ERRORS**

23699 No errors are defined.

23700 **EXAMPLES**

23701 None.

23702 **APPLICATION USAGE**23703 For a variable z of type **complex**:23704 `z == creal(z) + cimag(z)*I`23705 **RATIONALE**

23706 None.

23707 **FUTURE DIRECTIONS**

23708 None.

23709 **SEE ALSO**23710 *carg()*, *cimag()*, *conj()*, *cproj()*

23711 XBD &lt;complex.h&gt;

23712 **CHANGE HISTORY**

23713 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**creat()**23714 **NAME**23715 `creat` — create a new file or rewrite an existing one23716 **SYNOPSIS**23717 OH `#include <sys/stat.h>`23718 `#include <fcntl.h>`23719 `int creat(const char *path, mode_t mode);`23720 **DESCRIPTION**23721 The `creat()` function shall behave as if it is implemented as follows:23722 `int creat(const char *path, mode_t mode)`23723 `{`23724 `return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);`23725 `}`23726 **RETURN VALUE**23727 Refer to `open()`.23728 **ERRORS**23729 Refer to `open()`.23730 **EXAMPLES**23731 **Creating a File**23732 The following example creates the file `/tmp/file` with read and write permissions for the file  
23733 owner and read permission for group and others. The resulting file descriptor is assigned to the  
23734 `fd` variable.23735 `#include <fcntl.h>`23736 `...`23737 `int fd;`23738 `mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;`23739 `char *filename = "/tmp/file";`23740 `...`23741 `fd = creat(filename, mode);`23742 `...`23743 **APPLICATION USAGE**

23744 None.

23745 **RATIONALE**23746 The `creat()` function is redundant. Its services are also provided by the `open()` function. It has  
23747 been included primarily for historical purposes since many existing applications depend on it. It  
23748 is best considered a part of the C binding rather than a function that should be provided in other  
23749 languages.23750 **FUTURE DIRECTIONS**

23751 None.

23752 **SEE ALSO**23753 `mknod()`, `open()`23754 XBD `<fcntl.h>`, `<sys/stat.h>`, `<sys/types.h>`

23755 **CHANGE HISTORY**

23756 First released in Issue 1. Derived from Issue 1 of the SVID.

23757 **Issue 6**

23758 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

23759 The following new requirements on POSIX implementations derive from alignment with the  
23760 Single UNIX Specification:

- 23761 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
23762 required for conforming implementations of previous POSIX specifications, it was not  
23763 required for UNIX applications.

23764 **Issue 7**

23765 SD5-XSH-ERN-186 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**crypt()**23766 **NAME**23767 crypt — string encoding function (**CRYPT**)23768 **SYNOPSIS**

```
23769 xSI #include <unistd.h>
23770 char *crypt(const char *key, const char *salt);
```

23771 **DESCRIPTION**23772 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.23773 The *key* argument points to a string to be encoded. The *salt* argument shall be a string of at least  
23774 two bytes in length not including the null character chosen from the set:

```
23775 a b c d e f g h i j k l m n o p q r s t u v w x y z
23776 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
23777 0 1 2 3 4 5 6 7 8 9 . /
```

23778 The first two bytes of this string may be used to perturb the encoding algorithm.

23779 The return value of *crypt()* points to static data that is overwritten by each call.23780 The *crypt()* function need not be thread-safe.23781 **RETURN VALUE**23782 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two  
23783 bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null  
23784 pointer and set *errno* to indicate the error.23785 **ERRORS**23786 The *crypt()* function shall fail if:

23787 [ENOSYS] The functionality is not supported on this implementation.

23788 **EXAMPLES**23789 **Encoding Passwords**

23790 The following example finds a user database entry matching a particular user name and changes  
23791 the current password to a new password. The *crypt()* function generates an encoded version of  
23792 each password. The first call to *crypt()* produces an encoded version of the old password; that  
23793 encoded password is then compared to the password stored in the user database. The second  
23794 call to *crypt()* encodes the new password before it is stored.

23795 The *putpwent()* function, used in the following example, is not part of POSIX.1-2008.

```
23796 #include <unistd.h>
23797 #include <pwd.h>
23798 #include <string.h>
23799 #include <stdio.h>
23800 ...
23801 int valid_change;
23802 int pfd; /* Integer for file descriptor returned by open(). */
23803 FILE *fpfd; /* File pointer for use in putpwent(). */
23804 struct passwd *p;
23805 char user[100];
23806 char oldpasswd[100];
23807 char newpasswd[100];
23808 char savepasswd[100];
```

```

23809     ...
23810     valid_change = 0;
23811     while ((p = getpwent()) != NULL) {
23812         /* Change entry if found. */
23813         if (strcmp(p->pw_name, user) == 0) {
23814             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
23815                 strcpy(savepasswd, crypt(newpasswd, user));
23816                 p->pw_passwd = savepasswd;
23817                 valid_change = 1;
23818             }
23819             else {
23820                 fprintf(stderr, "Old password is not valid\n");
23821             }
23822         }
23823         /* Put passwd entry into ptmp. */
23824         putpwent(p, fpfd);
23825     }

```

**23826 APPLICATION USAGE**

23827 The values returned by this function need not be portable among XSI-conformant systems.

**23828 RATIONALE**

23829 None.

**23830 FUTURE DIRECTIONS**

23831 None.

**23832 SEE ALSO**

23833 *encrypt()*, *setkey()*

23834 XBD <unistd.h>

**23835 CHANGE HISTORY**

23836 First released in Issue 1. Derived from Issue 1 of the SVID.

**23837 Issue 5**

23838 Normative text previously in the APPLICATION USAGE section is moved to the  
23839 DESCRIPTION.

**23840 Issue 7**

23841 Austin Group Interpretation 1003.1-2001 #156 is applied.

23842 SD5-XSH-ERN-178 is applied.

**csin()**23843 **NAME**

23844 csin, csinf, csinl — complex sine functions

23845 **SYNOPSIS**

23846 #include &lt;complex.h&gt;

23847 double complex csin(double complex z);

23848 float complex csinf(float complex z);

23849 long double complex csinl(long double complex z);

23850 **DESCRIPTION**

23851 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23852 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23853 volume of POSIX.1-2008 defers to the ISO C standard.

23854 These functions shall compute the complex sine of z.

23855 **RETURN VALUE**

23856 These functions shall return the complex sine value.

23857 **ERRORS**

23858 No errors are defined.

23859 **EXAMPLES**

23860 None.

23861 **APPLICATION USAGE**

23862 None.

23863 **RATIONALE**

23864 None.

23865 **FUTURE DIRECTIONS**

23866 None.

23867 **SEE ALSO**23868 [casin\(\)](#)

23869 XBD &lt;complex.h&gt;

23870 **CHANGE HISTORY**

23871 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23872 **NAME**

23873           csinh, csinhf, csinhl — complex hyperbolic sine functions

23874 **SYNOPSIS**

23875           #include &lt;complex.h&gt;

23876           double complex csinh(double complex z);

23877           float complex csinhf(float complex z);

23878           long double complex csinhl(long double complex z);

23879 **DESCRIPTION**23880 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
23881 conflict between the requirements described here and the ISO C standard is unintentional. This  
23882 volume of POSIX.1-2008 defers to the ISO C standard.23883           These functions shall compute the complex hyperbolic sine of *z*.23884 **RETURN VALUE**

23885           These functions shall return the complex hyperbolic sine value.

23886 **ERRORS**

23887           No errors are defined.

23888 **EXAMPLES**

23889           None.

23890 **APPLICATION USAGE**

23891           None.

23892 **RATIONALE**

23893           None.

23894 **FUTURE DIRECTIONS**

23895           None.

23896 **SEE ALSO**23897           [casinh\(\)](#)

23898           XBD &lt;complex.h&gt;

23899 **CHANGE HISTORY**

23900           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**csinl()***System Interfaces*23901 **NAME**

23902           csinl — complex sine functions

23903 **SYNOPSIS**

23904           #include &lt;complex.h&gt;

23905           long double complex csinl(long double complex z);

23906 **DESCRIPTION**23907           Refer to *csin()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

23908 **NAME**

23909 csqrt, csqrtf, csqrtl — complex square root functions

23910 **SYNOPSIS**

```
23911 #include <complex.h>
23912 double complex csqrt(double complex z);
23913 float complex csqrtf(float complex z);
23914 long double complex csqrtl(long double complex z);
```

23915 **DESCRIPTION**

23916 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23917 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23918 volume of POSIX.1-2008 defers to the ISO C standard.

23919 These functions shall compute the complex square root of  $z$ , with a branch cut along the negative  
 23920 real axis.

23921 **RETURN VALUE**

23922 These functions shall return the complex square root value, in the range of the right half-plane  
 23923 (including the imaginary axis).

23924 **ERRORS**

23925 No errors are defined.

23926 **EXAMPLES**

23927 None.

23928 **APPLICATION USAGE**

23929 None.

23930 **RATIONALE**

23931 None.

23932 **FUTURE DIRECTIONS**

23933 None.

23934 **SEE ALSO**23935 [cabs\(\)](#), [cpow\(\)](#)23936 XBD [<complex.h>](#)23937 **CHANGE HISTORY**

23938 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**ctan()**23939 **NAME**23940 `ctan`, `ctanf`, `ctanl` — complex tangent functions23941 **SYNOPSIS**23942 `#include <complex.h>`23943 `double complex ctan(double complex z);`23944 `float complex ctanf(float complex z);`23945 `long double complex ctanl(long double complex z);`23946 **DESCRIPTION**23947 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
23948 conflict between the requirements described here and the ISO C standard is unintentional. This  
23949 volume of POSIX.1-2008 defers to the ISO C standard.23950 These functions shall compute the complex tangent of *z*.23951 **RETURN VALUE**

23952 These functions shall return the complex tangent value.

23953 **ERRORS**

23954 No errors are defined.

23955 **EXAMPLES**

23956 None.

23957 **APPLICATION USAGE**

23958 None.

23959 **RATIONALE**

23960 None.

23961 **FUTURE DIRECTIONS**

23962 None.

23963 **SEE ALSO**23964 [catan\(\)](#)23965 XBD [<complex.h>](#)23966 **CHANGE HISTORY**

23967 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23968 **NAME**

23969 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

23970 **SYNOPSIS**

23971 #include &lt;complex.h&gt;

23972 double complex ctanh(double complex z);

23973 float complex ctanhf(float complex z);

23974 long double complex ctanhl(long double complex z);

23975 **DESCRIPTION**23976 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
23977 conflict between the requirements described here and the ISO C standard is unintentional. This  
23978 volume of POSIX.1-2008 defers to the ISO C standard.23979 These functions shall compute the complex hyperbolic tangent of *z*.23980 **RETURN VALUE**

23981 These functions shall return the complex hyperbolic tangent value.

23982 **ERRORS**

23983 No errors are defined.

23984 **EXAMPLES**

23985 None.

23986 **APPLICATION USAGE**

23987 None.

23988 **RATIONALE**

23989 None.

23990 **FUTURE DIRECTIONS**

23991 None.

23992 **SEE ALSO**23993 [catanh\(\)](#)23994 XBD [<complex.h>](#)23995 **CHANGE HISTORY**

23996 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**ctanl()***System Interfaces*23997 **NAME**23998 `ctanl` — complex tangent functions23999 **SYNOPSIS**24000 `#include <complex.h>`24001 `long double complex ctanl(long double complex z);`24002 **DESCRIPTION**24003 Refer to *ctan()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

24004 **NAME**

24005 ctermid — generate a pathname for the controlling terminal

24006 **SYNOPSIS**

```
24007 CX #include <stdio.h>
24008 char *ctermid(char *s);
```

24009 **DESCRIPTION**

24010 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the  
 24011 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the  
 24012 file is not guaranteed.

24013 The *ctermid()* function need not be thread-safe if called with a NULL parameter.

24014 **RETURN VALUE**

24015 If *s* is a null pointer, the string shall be generated in an area that may be static (and therefore may  
 24016 be overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to  
 24017 point to a character array of at least `L_ctermid` bytes; the string is placed in this array and the  
 24018 value of *s* shall be returned. The symbolic constant `L_ctermid` is defined in `<stdio.h>`, and shall  
 24019 have a value greater than 0.

24020 The *ctermid()* function shall return an empty string if the pathname that would refer to the  
 24021 controlling terminal cannot be determined, or if the function is unsuccessful.

24022 **ERRORS**

24023 No errors are defined.

24024 **EXAMPLES**24025 **Determining the Controlling Terminal for the Current Process**

24026 The following example returns a pointer to a string that identifies the controlling terminal for the  
 24027 current process. The pathname for the terminal is stored in the array pointed to by the *ptr*  
 24028 argument, which has a size of `L_ctermid` bytes, as indicated by the *term* argument.

```
24029 #include <stdio.h>
24030 ...
24031 char term[L_ctermid];
24032 char *ptr;
24033 ptr = ctermid(term);
```

24034 **APPLICATION USAGE**

24035 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file  
 24036 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*  
 24037 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a  
 24038 pathname.

24039 **RATIONALE**

24040 `L_ctermid` must be defined appropriately for a given implementation and must be greater than  
 24041 zero so that array declarations using it are accepted by the compiler. The value includes the  
 24042 terminating null byte.

24043 Conforming applications that use multiple threads cannot call *ctermid()* with NULL as the  
 24044 parameter. If *s* is not NULL, the *ctermid()* function generates a string that, when used as a  
 24045 pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the  
 24046 return value of *ctermid()* is undefined.

**ctermid()**

24047 There is no additional burden on the programmer—changing to use a hypothetical thread-safe  
24048 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a  
24049 buffer. Application code should not assume that the returned string is short, as some  
24050 implementations have more than two pathname components before reaching a logical device  
24051 name.

**24052 FUTURE DIRECTIONS**

24053 None.

**24054 SEE ALSO**

24055 *ttyname()*

24056 XBD [<stdio.h>](#)

**24057 CHANGE HISTORY**

24058 First released in Issue 1. Derived from Issue 1 of the SVID.

**24059 Issue 5**

24060 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

**24061 Issue 6**

24062 The normative text is updated to avoid use of the term “must” for application requirements.

**24063 Issue 7**

24064 Austin Group Interpretation 1003.1-2001 #148 is applied, updating the RATIONALE.

24065 **NAME**

24066 ctime, ctime\_r — convert a time value to a date and time string

24067 **SYNOPSIS**

```
24068 OB #include <time.h>
24069 char *ctime(const time_t *clock);
24070 OB CX char *ctime_r(const time_t *clock, char *buf);
```

24071 **DESCRIPTION**

24072 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C  
 24073 standard. Any conflict between the requirements described here and the ISO C standard is  
 24074 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

24075 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds  
 24076 since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
24077 asctime(localtime(clock))
```

24078 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 24079 objects: a broken-down time structure and an array of **char**. Execution of any of the functions  
 24080 may overwrite the information returned in either of these objects by any of the other functions.

24081 The *ctime()* function need not be thread-safe.

24082 The *ctime\_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly  
 24083 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at  
 24084 least 26 bytes in size) and return *buf*.

24085 Unlike *ctime()*, the thread-safe version *ctime\_r()* is not required to set *tzname*.

24086 **RETURN VALUE**

24087 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time  
 24088 as an argument.

24089 CX Upon successful completion, *ctime\_r()* shall return a pointer to the string pointed to by *buf*.  
 24090 When an error is encountered, a null pointer shall be returned.

24091 **ERRORS**

24092 No errors are defined.

24093 **EXAMPLES**

24094 None.

24095 **APPLICATION USAGE**

24096 These functions are included only for compatibility with older implementations. They have  
 24097 undefined behavior if the resulting string would be too long, so the use of these functions  
 24098 should be discouraged. On implementations that do not detect output string length overflow, it  
 24099 is possible to overflow the output buffers in such a way as to cause applications to fail, or  
 24100 possible system security violations. Also, these functions do not support localized date and time  
 24101 formats. To avoid these problems, applications should use *strftime()* to generate strings from  
 24102 broken-down times.

24103 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

24104 The *ctime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead of  
 24105 possibly using a static data area that may be overwritten by each call.

24106 Attempts to use *ctime()* or *ctime\_r()* for times before the Epoch or for times beyond the year 9999  
 24107 produce undefined results. Refer to *asctime()* (on page 590).

**ctime()**24108 **RATIONALE**

24109 The standard developers decided to mark the *ctime()* and *ctime\_r()* functions obsolescent even  
 24110 though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C  
 24111 standard also provides the *strftime()* function which can be used to avoid these problems.

24112 **FUTURE DIRECTIONS**

24113 These functions may be removed in a future version.

24114 **SEE ALSO**

24115 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*

24116 XBD <[time.h](#)>

24117 **CHANGE HISTORY**

24118 First released in Issue 1. Derived from Issue 1 of the SVID.

24119 **Issue 5**

24120 Normative text previously in the APPLICATION USAGE section is moved to the  
 24121 DESCRIPTION.

24122 The *ctime\_r()* function is included for alignment with the POSIX Threads Extension.

24123 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.

24124 **Issue 6**

24125 Extensions beyond the ISO C standard are marked.

24126 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24127 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 24128 its avoidance of possibly using a static data area.

24129 **Issue 7**

24130 Austin Group Interpretation 1003.1-2001 #156 is applied.

24131 SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

24132 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

24133 The *ctime\_r()* function is moved from the Thread-Safe Functions option to the Base.

24134 **NAME**

24135 daylight — daylight savings time flag

24136 **SYNOPSIS**

```
24137 XSI #include <time.h>  
24138 extern int daylight;
```

24139 **DESCRIPTION**24140 Refer to [tzset\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**dbm\_clearerr()**24141 **NAME**

24142 dbm\_clearerr, dbm\_close, dbm\_delete, dbm\_error, dbm\_fetch, dbm\_firstkey, dbm\_nextkey,  
24143 dbm\_open, dbm\_store — database functions

24144 **SYNOPSIS**

```
24145 xSI #include <ndbm.h>
24146 int dbm_clearerr(DBM *db);
24147 void dbm_close(DBM *db);
24148 int dbm_delete(DBM *db, datum key);
24149 int dbm_error(DBM *db);
24150 datum dbm_fetch(DBM *db, datum key);
24151 datum dbm_firstkey(DBM *db);
24152 datum dbm_nextkey(DBM *db);
24153 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
24154 int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

24155 **DESCRIPTION**

24156 These functions create, access, and modify a database.

24157 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object  
24158 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in  
24159 the object pointed to by *dptr*.

24160 A database shall be stored in one or two files. When one file is used, the name of the database  
24161 file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm\_open()*. When  
24162 two files are used, the names of the database files shall be formed by appending the suffixes **.dir**  
24163 and **.pag** respectively to the *file* argument.

24164 The *dbm\_open()* function shall open a database. The *file* argument to the function is the  
24165 pathname of the database. The *open\_flags* argument has the same meaning as the *flags* argument  
24166 of *open()* except that a database opened for write-only access opens the files for read and write  
24167 access and the behavior of the **O\_APPEND** flag is unspecified. The *file\_mode* argument has the  
24168 same meaning as the third argument of *open()*.

24169 The *dbm\_open()* function need not accept pathnames longer than  $\{\text{PATH\_MAX}\}-4$  bytes  
24170 (including the terminating null), or pathnames with a last component longer than  
24171  $\{\text{NAME\_MAX}\}-4$  bytes (excluding the terminating null).

24172 The *dbm\_close()* function shall close a database. The application shall ensure that argument *db* is  
24173 a pointer to a **dbm** structure that has been returned from a call to *dbm\_open()*.

24174 These database functions shall support an internal block size large enough to support  
24175 key/content pairs of at least 1 023 bytes.

24176 The *dbm\_fetch()* function shall read a record from a database. The argument *db* is a pointer to a  
24177 database structure that has been returned from a call to *dbm\_open()*. The argument *key* is a  
24178 **datum** that has been initialized by the application to the value of the key that matches the key of  
24179 the record the program is fetching.

24180 The *dbm\_store()* function shall write a record to a database. The argument *db* is a pointer to a  
24181 database structure that has been returned from a call to *dbm\_open()*. The argument *key* is a  
24182 **datum** that has been initialized by the application to the value of the key that identifies (for  
24183 subsequent reading, writing, or deleting) the record the application is writing. The argument  
24184 *content* is a **datum** that has been initialized by the application to the value of the record the  
24185 program is writing. The argument *store\_mode* controls whether *dbm\_store()* replaces any pre-  
24186 existing record that has the same key that is specified by the *key* argument. The application shall

24187 set *store\_mode* to either DBM\_INSERT or DBM\_REPLACE. If the database contains a record that  
 24188 matches the *key* argument and *store\_mode* is DBM\_REPLACE, the existing record shall be  
 24189 replaced with the new record. If the database contains a record that matches the *key* argument  
 24190 and *store\_mode* is DBM\_INSERT, the existing record shall be left unchanged and the new record  
 24191 ignored. If the database does not contain a record that matches the *key* argument and *store\_mode*  
 24192 is either DBM\_INSERT or DBM\_REPLACE, the new record shall be inserted in the database.

24193 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.  
 24194 Moreover, the application shall ensure that all key/content pairs that hash together fit on a  
 24195 single block. The *dbm\_store()* function shall return an error in the event that a disk block fills  
 24196 with inseparable data.

24197 The *dbm\_delete()* function shall delete a record and its key from the database. The argument *db* is  
 24198 a pointer to a database structure that has been returned from a call to *dbm\_open()*. The argument  
 24199 *key* is a **datum** that has been initialized by the application to the value of the key that identifies  
 24200 the record the program is deleting.

24201 The *dbm\_firstkey()* function shall return the first key in the database. The argument *db* is a  
 24202 pointer to a database structure that has been returned from a call to *dbm\_open()*.

24203 The *dbm\_nextkey()* function shall return the next key in the database. The argument *db* is a  
 24204 pointer to a database structure that has been returned from a call to *dbm\_open()*. The application  
 24205 shall ensure that the *dbm\_firstkey()* function is called before calling *dbm\_nextkey()*. Subsequent  
 24206 calls to *dbm\_nextkey()* return the next key until all of the keys in the database have been  
 24207 returned.

24208 The *dbm\_error()* function shall return the error condition of the database. The argument *db* is a  
 24209 pointer to a database structure that has been returned from a call to *dbm\_open()*.

24210 The *dbm\_clearerr()* function shall clear the error condition of the database. The argument *db* is a  
 24211 pointer to a database structure that has been returned from a call to *dbm\_open()*.

24212 The *dptr* pointers returned by these functions may point into static storage that may be changed  
 24213 by subsequent calls.

24214 These functions need not be thread-safe.

#### 24215 RETURN VALUE

24216 The *dbm\_store()* and *dbm\_delete()* functions shall return 0 when they succeed and a negative  
 24217 value when they fail.

24218 The *dbm\_store()* function shall return 1 if it is called with a *flags* value of DBM\_INSERT and the  
 24219 function finds an existing record with the same key.

24220 The *dbm\_error()* function shall return 0 if the error condition is not set and return a non-zero  
 24221 value if the error condition is set.

24222 The return value of *dbm\_clearerr()* is unspecified.

24223 The *dbm\_firstkey()* and *dbm\_nextkey()* functions shall return a key **datum**. When the end of the  
 24224 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*  
 24225 member of the key shall be a null pointer and the error condition of the database shall be set.

24226 The *dbm\_fetch()* function shall return a content **datum**. If no record in the database matches the  
 24227 key or if an error condition has been detected in the database, the *dptr* member of the content  
 24228 shall be a null pointer.

24229 The *dbm\_open()* function shall return a pointer to a database structure. If an error is detected  
 24230 during the operation, *dbm\_open()* shall return a **(DBM \*)0**.

**dbm\_clearerr()**24231 **ERRORS**

24232 No errors are defined.

24233 **EXAMPLES**

24234 None.

24235 **APPLICATION USAGE**

24236 The following code can be used to traverse the database:

24237 

```
for(key = dbm_firstkey(db); key.dpstr != NULL; key = dbm_nextkey(db))
```

24238 The *dbm\_\** functions provided in this library should not be confused in any way with those of a  
 24239 general-purpose database management system. These functions do not provide for multiple  
 24240 search keys per entry, they do not protect against multi-user access (in other words they do not  
 24241 lock records or files), and they do not provide the many other useful database functions that are  
 24242 found in more robust database management systems. Creating and updating databases by use of  
 24243 these functions is relatively slow because of data copies that occur upon hash collisions. These  
 24244 functions are useful for applications requiring fast lookup of relatively static information that is  
 24245 to be indexed by a single key.

24246 Note that a strictly conforming application is extremely limited by these functions: since there is  
 24247 no way to determine that the keys in use do not all hash to the same value (although that would  
 24248 be rare), a strictly conforming application cannot be guaranteed that it can store more than one  
 24249 block's worth of data in the database. As long as a key collision does not occur, additional data  
 24250 may be stored, but because there is no way to determine whether an error is due to a key  
 24251 collision or some other error condition (*dbm\_error()* being effectively a Boolean), once an error is  
 24252 detected, the application is effectively limited to guessing what the error might be if it wishes to  
 24253 continue using these functions.

24254 The *dbm\_delete()* function need not physically reclaim file space, although it does make it  
 24255 available for reuse by the database.

24256 After calling *dbm\_store()* or *dbm\_delete()* during a pass through the keys by *dbm\_firstkey()* and  
 24257 *dbm\_nextkey()*, the application should reset the database by calling *dbm\_firstkey()* before again  
 24258 calling *dbm\_nextkey()*. The contents of these files are unspecified and may not be portable.

24259 Applications should take care that database pathname arguments specified to *dbm\_open()* are  
 24260 not prefixes of unrelated files. This might be done, for example, by placing databases in a  
 24261 separate directory.

24262 Since some implementations use three characters for a suffix and others use four characters for a  
 24263 suffix, applications should ensure that the maximum portable pathname length passed to  
 24264 *dbm\_open()* is no greater than {PATH\_MAX}-4 bytes, with the last component of the pathname  
 24265 no greater than {NAME\_MAX}-4 bytes.

24266 **RATIONALE**

24267 Previously the standard required the database to be stored in two files, one file being a directory  
 24268 containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and  
 24269 having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow  
 24270 newer implementations of the Berkeley DB interface using a single file that have evolved while  
 24271 remaining compatible with the application programming interface. The standard developers  
 24272 considered removing the specific suffixes altogether but decided to retain them so as not to  
 24273 pollute the application file name space more than necessary and to allow for portable backups of  
 24274 the database.

24275 **FUTURE DIRECTIONS**

24276 None.

24277 **SEE ALSO**24278 *open()*

24279 XBD &lt;ndbm.h&gt;

24280 **CHANGE HISTORY**

24281 First released in Issue 4, Version 2.

24282 **Issue 5**

24283 Moved from X/OPEN UNIX extension to BASE.

24284 Normative text previously in the APPLICATION USAGE section is moved to the  
24285 DESCRIPTION.

24286 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

24287 **Issue 6**

24288 The normative text is updated to avoid use of the term “must” for application requirements.

24289 **Issue 7**24290 Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits  
24291 newer implementations of the Berkeley DB interface.

24292 Austin Group Interpretation 1003.1-2001 #156 is applied.

**difftime()**24293 **NAME**24294 `difftime` — compute the difference between two calendar time values24295 **SYNOPSIS**24296 `#include <time.h>`24297 `double difftime(time_t time1, time_t time0);`24298 **DESCRIPTION**24299 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24300 conflict between the requirements described here and the ISO C standard is unintentional. This  
24301 volume of POSIX.1-2008 defers to the ISO C standard.24302 The `difftime()` function shall compute the difference between two calendar times (as returned by  
24303 `time()`):  $time1 - time0$ .24304 **RETURN VALUE**24305 The `difftime()` function shall return the difference expressed in seconds as a type **double**.24306 **ERRORS**

24307 No errors are defined.

24308 **EXAMPLES**

24309 None.

24310 **APPLICATION USAGE**

24311 None.

24312 **RATIONALE**

24313 None.

24314 **FUTURE DIRECTIONS**

24315 None.

24316 **SEE ALSO**24317 `asctime()`, `clock()`, `ctime()`, `gmtime()`, `localtime()`, `mktime()`, `strftime()`, `strptime()`, `time()`, `utime()`24318 XBD `<time.h>`24319 **CHANGE HISTORY**

24320 First released in Issue 4. Derived from the ISO C standard.

24321 **NAME**

24322 dirfd — extract the file descriptor used by a DIR stream

24323 **SYNOPSIS**

```
24324 #include <dirent.h>
24325 int dirfd(DIR *dirp);
```

24326 **DESCRIPTION**

24327 The *dirfd()* function shall return a file descriptor referring to the same directory as the *dirp* argument. This file descriptor shall be closed by a call to *closedir()*. If any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir()*, *readdir()*, *readdir\_r()*, or *rewinddir()*, the behavior is undefined.

24331 **RETURN VALUE**

24332 Upon successful completion, the *dirfd()* function shall return an integer which contains a file descriptor for the stream pointed to by *dirp*. Otherwise, it shall return  $-1$  and may set *errno* to indicate the error.

24335 **ERRORS**24336 The *dirfd()* function may fail if:

- |       |           |                                                                                            |
|-------|-----------|--------------------------------------------------------------------------------------------|
| 24337 | [EINVAL]  | The <i>dirp</i> argument does not refer to a valid directory stream.                       |
| 24338 | [ENOTSUP] | The implementation does not support the association of a file descriptor with a directory. |

24340 **EXAMPLES**

24341 None.

24342 **APPLICATION USAGE**

24343 The *dirfd()* function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir()* function.

24345 **RATIONALE**

24346 This interface was introduced because the Base Definitions volume of POSIX.1-2008 does not make public the **DIR** data structure. Applications tend to use the *fchdir()* function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

24350 The description uses the term “a file descriptor” rather than “the file descriptor”. The implication intended is that an implementation that does not use an *fd* for *diropen()* could still *open()* the directory to implement the *dirfd()* function. Such a descriptor must be closed later during a call to *closedir()*.

24354 An implementation that does not support file descriptors referring to directories may fail with [ENOTSUP].

24356 If it is necessary to allocate an *fd* to be returned by *dirfd()*, it should be done at the time of a call to *opendir()*.

24358 **FUTURE DIRECTIONS**

24359 None.

24360 **SEE ALSO**24361 *closedir()*, *fchdir()*, *fdopendir()*, *fileno()*, *open()*, *readdir()*24362 XBD [<dirent.h>](#)

## dirfd()

|       |                            |
|-------|----------------------------|
| 24363 | <b>CHANGE HISTORY</b>      |
| 24364 | First released in Issue 7. |

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

24365 **NAME**

24366            dirname — report the parent directory name of a file pathname

24367 **SYNOPSIS**

```
24368 XSI      #include <libgen.h>
24369         char *dirname(char *path);
```

24370 **DESCRIPTION**

24371        The *dirname()* function shall take a pointer to a character string that contains a pathname, and  
 24372        return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/'  
 24373        characters in the path are not counted as part of the path.

24374        If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a  
 24375        null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".

24376        The *dirname()* function need not be thread-safe.

24377 **RETURN VALUE**

24378        The *dirname()* function shall return a pointer to a string that is the parent directory of *path*. If  
 24379        *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.

24380        The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to  
 24381        static storage that may then be overwritten by subsequent calls to *dirname()*.

24382 **ERRORS**

24383        No errors are defined.

24384 **EXAMPLES**

24385        The following code fragment reads a pathname, changes the current working directory to the  
 24386        parent directory, and opens the file.

```
24387     char *path = NULL, *pathcopy;
24388     size_t buflen = 0;
24389     ssize_t linelen = 0;
24390     int fd;
24391
24392     linelen = getline(&path, &buflen, stdin);
24393     path[linelen-1] = 0;
24394     pathcopy = strdup(path);
24395     if (chdir(dirname(pathcopy)) < 0) {
24396         ...
24397     }
24398     if ((fd = open(basename(path), O_RDONLY)) >= 0) {
24399         ...
24400         close (fd);
24401     }
24402     ...
24403     free (pathcopy);
24404     free (path);
```

# dirname()

24404 **Sample Input and Output Strings for dirname()**

24405 In the following table, the input string is the value pointed to by *path*, and the output string is  
 24406 the return value of the *dirname()* function.

| Input String | Output String |
|--------------|---------------|
| "/usr/lib"   | "/usr"        |
| "/usr/"      | "/"           |
| "usr"        | "."           |
| "/"          | "/"           |
| "."          | "."           |
| "."          | "."           |

24414 **APPLICATION USAGE**

24415 The *dirname()* and *basename()* functions together yield a complete pathname. The expression  
 24416 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

24417 Since the meaning of the leading *"/"* is implementation-defined, *dirname("//foo)* may return  
 24418 either *"/"* or *'/'* (but nothing else).

24419 **RATIONALE**

24420 None.

24421 **FUTURE DIRECTIONS**

24422 None.

24423 **SEE ALSO**

24424 *basename()*

24425 XBD <libgen.h>

24426 **CHANGE HISTORY**

24427 First released in Issue 4, Version 2.

24428 **Issue 5**

24429 Moved from X/OPEN UNIX extension to BASE.

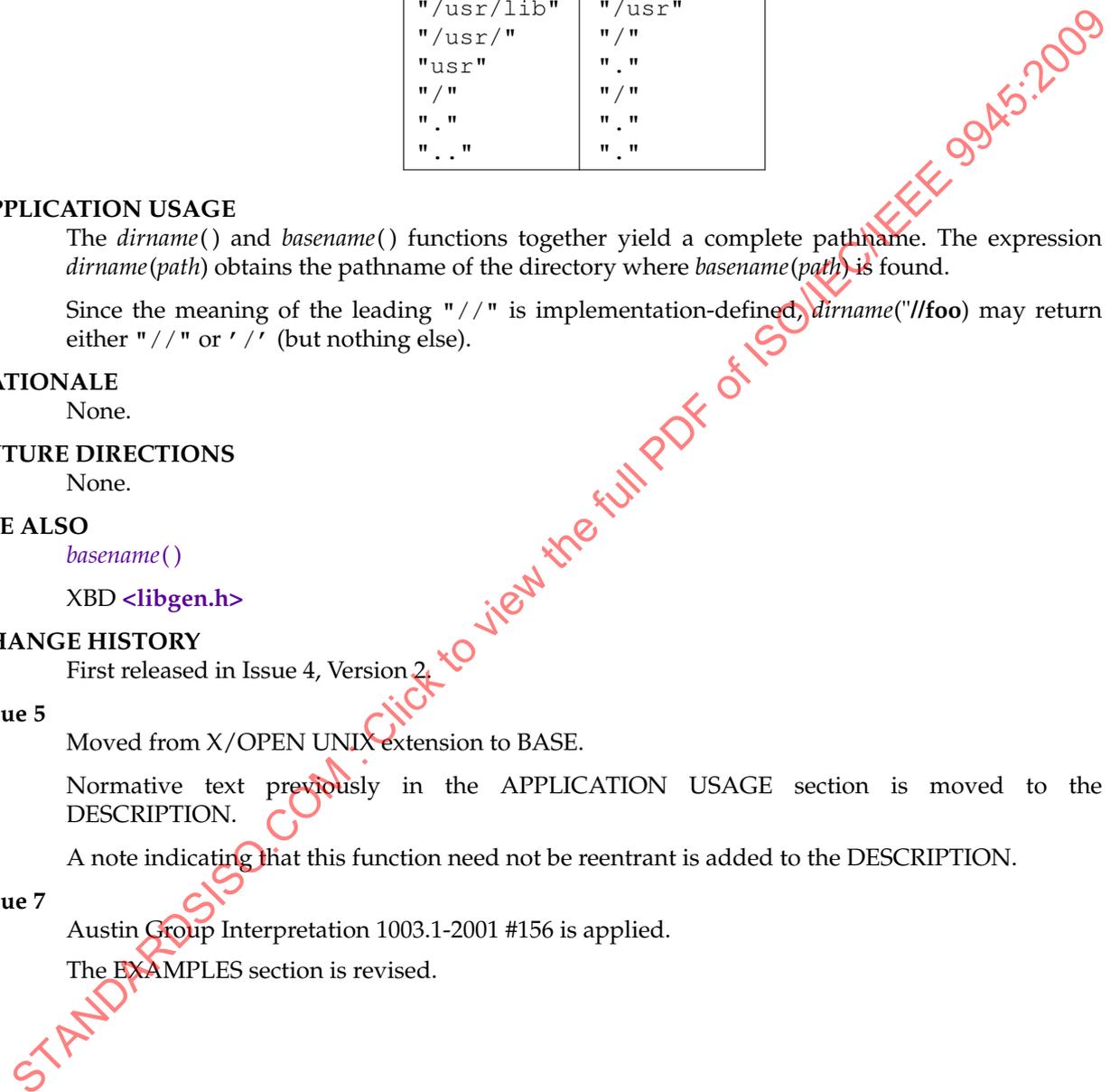
24430 Normative text previously in the APPLICATION USAGE section is moved to the  
 24431 DESCRIPTION.

24432 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

24433 **Issue 7**

24434 Austin Group Interpretation 1003.1-2001 #156 is applied.

24435 The EXAMPLES section is revised.



24436 **NAME**

24437 div — compute the quotient and remainder of an integer division

24438 **SYNOPSIS**

24439 #include &lt;stdlib.h&gt;

24440 div\_t div(int *numer*, int *denom*);24441 **DESCRIPTION**

24442 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24443 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24444 volume of POSIX.1-2008 defers to the ISO C standard.

24445 The *div()* function shall compute the quotient and remainder of the division of the numerator  
 24446 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer  
 24447 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be  
 24448 represented, the behavior is undefined; otherwise,  $quot * denom + rem$  shall equal *numer*.

24449 **RETURN VALUE**

24450 The *div()* function shall return a structure of type **div\_t**, comprising both the quotient and the  
 24451 remainder. The structure includes the following members, in any order:

```
24452 int quot; /* quotient */
24453 int rem; /* remainder */
```

24454 **ERRORS**

24455 No errors are defined.

24456 **EXAMPLES**

24457 None.

24458 **APPLICATION USAGE**

24459 None.

24460 **RATIONALE**

24461 None.

24462 **FUTURE DIRECTIONS**

24463 None.

24464 **SEE ALSO**24465 [ldiv\(\)](#)24466 XBD [<stdlib.h>](#)24467 **CHANGE HISTORY**

24468 First released in Issue 4. Derived from the ISO C standard.

**dlclose()**24469 **NAME**24470 `dlclose` — close a `dlopen()` object24471 **SYNOPSIS**

```
24472 #include <dlfcn.h>
24473 int dlclos(void *handle);
```

24474 **DESCRIPTION**

24475 The `dlclose()` function shall inform the system that the object referenced by a `handle` returned  
 24476 from a previous `dlopen()` invocation is no longer needed by the application.

24477 The use of `dlclose()` reflects a statement of intent on the part of the process, but does not create  
 24478 any requirement upon the implementation, such as removal of the code or symbols referenced  
 24479 by `handle`. Once an object has been closed using `dlclose()` an application should assume that its  
 24480 symbols are no longer available to `dlsym()`. All objects loaded automatically as a result of  
 24481 invoking `dlopen()` on the referenced object shall also be closed if this is the last reference to it.

24482 Although a `dlclose()` operation is not required to remove structures from an address space,  
 24483 neither is an implementation prohibited from doing so. The only restriction on such a removal is  
 24484 that no object shall be removed to which references have been relocated, until or unless all such  
 24485 references are removed. For instance, an object that had been loaded with a `dlopen()` operation  
 24486 specifying the `RTLD_GLOBAL` flag might provide a target for dynamic relocations performed in  
 24487 the processing of other objects—in such environments, an application may assume that no  
 24488 relocation, once made, shall be undone or remade unless the object requiring the relocation has  
 24489 itself been removed.

24490 **RETURN VALUE**

24491 If the referenced object was successfully closed, `dlclose()` shall return 0. If the object could not be  
 24492 closed, or if `handle` does not refer to an open object, `dlclose()` shall return a non-zero value. More  
 24493 detailed diagnostic information shall be available through `dLError()`.

24494 **ERRORS**

24495 No errors are defined.

24496 **EXAMPLES**24497 The following example illustrates use of `dlopen()` and `dlclose()`:

```
24498 ...
24499 /* Open a dynamic library and then close it ... */
24500 #include <dlfcn.h>
24501 void *mylib;
24502 int eret;
24503
24504 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
24505 eret = dlclos(mylib);
24506 ...
```

24507 **APPLICATION USAGE**

24508 A conforming application should employ a `handle` returned from a `dlopen()` invocation only  
 24509 within a given scope bracketed by the `dlopen()` and `dlclose()` operations. Implementations are  
 24510 free to use reference counting or other techniques such that multiple calls to `dlopen()` referencing  
 24511 the same object may return the same object for `handle`. Implementations are also free to reuse a  
 24512 `handle`. For these reasons, the value of a `handle` must be treated as an opaque object by the  
 24513 application, used only in calls to `dlsym()` and `dlclose()`.

24514 **RATIONALE**

24515 None.

24516 **FUTURE DIRECTIONS**

24517 None.

24518 **SEE ALSO**24519 *dLError()*, *dlopen()*, *dlsym()*

24520 XBD &lt;dlfcn.h&gt;

24521 **CHANGE HISTORY**

24522 First released in Issue 5.

24523 **Issue 6**24524 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last  
24525 reference to it”.24526 **Issue 7**24527 The *dlopen()* function is moved from the XSI option to Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**dlerror()**24528 **NAME**24529 `dlerror` — get diagnostic information24530 **SYNOPSIS**

```
24531 #include <dlfcn.h>
24532 char *dlerror(void);
```

24533 **DESCRIPTION**

24534 The `dlerror()` function shall return a null-terminated character string (with no trailing  
 24535 <newline>) that describes the last error that occurred during dynamic linking processing. If no  
 24536 dynamic linking errors have occurred since the last invocation of `dlerror()`, `dlerror()` shall return  
 24537 NULL. Thus, invoking `dlerror()` a second time, immediately following a prior invocation, shall  
 24538 result in NULL being returned.

24539 The `dlerror()` function need not be thread-safe.

24540 **RETURN VALUE**

24541 If successful, `dlerror()` shall return a null-terminated character string; otherwise, NULL shall be  
 24542 returned.

24543 **ERRORS**

24544 No errors are defined.

24545 **EXAMPLES**

24546 The following example prints out the last dynamic linking error:

```
24547 ...
24548 #include <dlfcn.h>
24549 char *errstr;
24550 errstr = dlerror();
24551 if (errstr != NULL)
24552     printf ("A dynamic linking error occurred: (%s)\n", errstr);
24553 ...
```

24554 **APPLICATION USAGE**

24555 The messages returned by `dlerror()` may reside in a static buffer that is overwritten on each call  
 24556 to `dlerror()`. Application code should not write to this buffer. Programs wishing to preserve an  
 24557 error message should make their own copies of that message. Depending on the application  
 24558 environment with respect to asynchronous execution events, such as signals or other  
 24559 asynchronous computation sharing the address space, conforming applications should use a  
 24560 critical section to retrieve the error pointer and buffer.

24561 **RATIONALE**

24562 None.

24563 **FUTURE DIRECTIONS**

24564 None.

24565 **SEE ALSO**

24566 [\*dlclose\(\)\*](#), [\*dlopen\(\)\*](#), [\*dlsym\(\)\*](#)

24567 XBD [\*<dlfcn.h>\*](#)

24568 **CHANGE HISTORY**

24569 First released in Issue 5.

- 24570 **Issue 6**  
24571 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 24572 **Issue 7**  
24573 Austin Group Interpretation 1003.1-2001 #156 is applied.  
24574 The *dlerror()* function is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**dlopen()**24575 **NAME**24576 `dlopen` — gain access to an executable object file24577 **SYNOPSIS**24578 `#include <dlfcn.h>`24579 `void *dlopen(const char *file, int mode);`24580 **DESCRIPTION**

24581 The `dlopen()` function shall make an executable object file specified by *file* available to the calling  
 24582 program. The class of files eligible for this operation and the manner of their construction are  
 24583 implementation-defined, though typically such files are executable objects such as shared  
 24584 libraries, relocatable files, or programs. Note that some implementations permit the construction  
 24585 of dependencies between such objects that are embedded within files. In such cases, a `dlopen()`  
 24586 operation shall load such dependencies in addition to the object referenced by *file*.  
 24587 Implementations may also impose specific constraints on the construction of programs that can  
 24588 employ `dlopen()` and its related services.

24589 A successful `dlopen()` shall return a *handle* which the caller may use on subsequent calls to  
 24590 `dlsym()` and `dlclose()`. The value of this *handle* should not be interpreted in any way by the caller.

24591 The *file* argument is used to construct a pathname to the object file. If *file* contains a <slash>  
 24592 character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an  
 24593 implementation-defined manner to yield a pathname.

24594 If the value of *file* is 0, `dlopen()` shall provide a *handle* on a global symbol object. This object shall  
 24595 provide access to the symbols from an ordered set of objects consisting of the original program  
 24596 image file, together with any objects loaded at program start-up as specified by that process  
 24597 image file (for example, shared libraries), and the set of objects loaded using a `dlopen()` operation  
 24598 together with the `RTLD_GLOBAL` flag. As the latter set of objects can change during execution,  
 24599 the set identified by *handle* can also change dynamically.

24600 Only a single copy of an object file is brought into the address space, even if `dlopen()` is invoked  
 24601 multiple times in reference to the file, and even if different pathnames are used to reference the  
 24602 file.

24603 The *mode* parameter describes how `dlopen()` shall operate upon *file* with respect to the processing  
 24604 of relocations and the scope of visibility of the symbols provided within *file*. When an object is  
 24605 brought into the address space of a process, it may contain references to symbols whose  
 24606 addresses are not known until the object is loaded. These references shall be relocated before the  
 24607 symbols can be accessed. The *mode* parameter governs when these relocations take place and  
 24608 may have the following values:

|       |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 24609 | <code>RTLD_LAZY</code> | Relocations shall be performed at an implementation-defined time,<br>ranging from the time of the <code>dlopen()</code> call until the first reference to a<br>given symbol occurs. Specifying <code>RTLD_LAZY</code> should improve<br>performance on implementations supporting dynamic symbol binding as<br>a process may not reference all of the functions in any given object. And,<br>for systems supporting dynamic symbol resolution for normal process<br>execution, this behavior mimics the normal handling of process<br>execution. |
|-------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|       |                       |                                                                                                                                                                                                                                                                                                                                                          |
|-------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 24617 | <code>RTLD_NOW</code> | All necessary relocations shall be performed when the object is first<br>loaded. This may waste some processing if relocations are performed for<br>functions that are never referenced. This behavior may be useful for<br>applications that need to know as soon as an object is loaded that all<br>symbols referenced during execution are available. |
|-------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

24622 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the  
 24623 symbols in the original process image file, any objects loaded at program start-up, from the  
 24624 object itself as well as any other object included in the same *dlopen()* invocation, and any objects  
 24625 that were loaded in any *dlopen()* invocation and which specified the RTLD\_GLOBAL flag. To  
 24626 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*  
 24627 parameter should be a bitwise-inclusive OR with one of the following values:

24628 RTLD\_GLOBAL The object's symbols shall be made available for the relocation processing  
 24629 of any other object. In addition, symbol lookup using *dlopen(0, mode)* and  
 24630 an associated *dlsym()* allows objects loaded with this *mode* to be searched.

24631 RTLD\_LOCAL The object's symbols shall not be made available for the relocation  
 24632 processing of any other object.

24633 If neither RTLD\_GLOBAL nor RTLD\_LOCAL are specified, then the default behavior is  
 24634 unspecified.

24635 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,  
 24636 however, that once RTLD\_NOW has been specified all relocations shall have been completed  
 24637 rendering further RTLD\_NOW operations redundant and any further RTLD\_LAZY operations  
 24638 irrelevant. Similarly, note that once RTLD\_GLOBAL has been specified the object shall maintain  
 24639 the RTLD\_GLOBAL status regardless of any previous or future specification of RTLD\_LOCAL,  
 24640 as long as the object remains in the address space (see *dlclose()*).

24641 Symbols introduced into a program through calls to *dlopen()* may be used in relocation activities.  
 24642 Symbols so introduced may duplicate symbols already defined by the program or previous  
 24643 *dlopen()* operations. To resolve the ambiguities such a situation might present, the resolution of a  
 24644 symbol reference to symbol definition is based on a symbol resolution order. Two such  
 24645 resolution orders are defined: *load* or *dependency* ordering. Load order establishes an ordering  
 24646 among symbol definitions, such that the definition first loaded (including definitions from the  
 24647 image file and any dependent objects loaded with it) has priority over objects added later (via  
 24648 *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a breadth-  
 24649 first order starting with a given object, then all of its dependencies, then any dependents of  
 24650 those, iterating until all dependencies are satisfied. With the exception of the global symbol  
 24651 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the  
 24652 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

24653 When an object is first made accessible via *dlopen()* it and its dependent objects are added in  
 24654 dependency order. Once all the objects are added, relocations are performed using load order.  
 24655 Note that if an object or its dependencies had been previously loaded, the load and dependency  
 24656 orders may yield different resolutions.

24657 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum  
 24658 those which are exported as symbols of global scope by the object. Typically such symbols shall  
 24659 be those that were specified in (for example) C source code as having *extern* linkage. The precise  
 24660 manner in which an implementation constructs the set of exported symbols for a *dlopen()* object  
 24661 is specified by that implementation.

#### 24662 RETURN VALUE

24663 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for  
 24664 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its  
 24665 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be  
 24666 available through *dlerror()*.

**dlopen()**24667 **ERRORS**

24668 No errors are defined.

24669 **EXAMPLES**24670 Refer to *dlsym()* (on page 735).24671 **APPLICATION USAGE**

24672 None.

24673 **RATIONALE**

24674 None.

24675 **FUTURE DIRECTIONS**

24676 None.

24677 **SEE ALSO**24678 *dlclose()*, *dlderror()*, *dlsym()*24679 XBD **<dlfcn.h>**24680 **CHANGE HISTORY**

24681 First released in Issue 5.

24682 **Issue 6**24683 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default  
24684 behavior in the DESCRIPTION when neither RTLD\_GLOBAL nor RTLD\_LOCAL are specified  
24685 from implementation-defined to unspecified.24686 **Issue 7**24687 The *dlopen()* function is moved from the XSI option to the Base.24688 The EXAMPLES section is updated to refer to *dlsym()*.

24689 **NAME**24690 dlsym — obtain the address of a symbol from a *dlopen()* object24691 **SYNOPSIS**

24692 #include &lt;dlfcn.h&gt;

24693 void \*dlsym(void \*restrict handle, const char \*restrict name);

24694 **DESCRIPTION**

24695 The *dlsym()* function shall obtain the address of a symbol defined within an object made  
 24696 accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to  
 24697 *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's  
 24698 name as a character string.

24699 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a  
 24700 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()*  
 24701 operations upon the global symbol object. The symbol resolution algorithm used shall be  
 24702 dependency order as described in *dlopen()*.

24703 The RTLD\_DEFAULT and RTLD\_NEXT flags are reserved for future use.

24704 **RETURN VALUE**

24705 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be  
 24706 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More  
 24707 detailed diagnostic information shall be available through *dLError()*.

24708 **ERRORS**

24709 No errors are defined.

24710 **EXAMPLES**

24711 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or  
 24712 data objects. For simplicity, error checking has been omitted.

```
24713 void    *handle;
24714 int     *iptr, (*fptr)(int);

24715 /* open the needed object */
24716 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

24717 /* find the address of function and data objects */
24718 *(void **)(&fptr) = dlsym(handle, "my_function");
24719 iptr = (int *)dlsym(handle, "my_object");

24720 /* invoke function, passing value of integer as a parameter */
24721 (*fptr)(*iptr);
```

24722 **APPLICATION USAGE**

24723 Special purpose values for *handle* are reserved for future use. These values and their meanings  
 24724 are:

24725 RTLD\_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a  
 24726 symbol using this handle would find the same definition as a direct use of this  
 24727 symbol in the program code.

24728 RTLD\_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the  
 24729 object containing the invocation of *dlsym()*. The *next* object is the one found  
 24730 upon the application of a load order symbol resolution algorithm (see  
 24731 *dlopen()*). The next object is either one of global scope (because it was  
 24732 introduced as part of the original process image or because it was added with  
 24733 a *dlopen()* operation including the RTLD\_GLOBAL flag), or is an object that

**dlsym()**

24734 was included in the same *dlopen()* operation that loaded this one.

24735 The RTLD\_NEXT flag is useful to navigate an intentionally created hierarchy

24736 of multiply-defined symbols created through *interposition*. For example, if a

24737 program wished to create an implementation of *malloc()* that embedded some

24738 statistics gathering about memory allocations, such an implementation could

24739 use the real *malloc()* definition to perform the memory allocation—and itself

24740 only embed the necessary logic to implement the statistics gathering function.

**RATIONALE**

24741 The ISO C standard does not require that pointers to functions can be cast back and forth to

24742 pointers to data. However, POSIX-conforming implementations are required to support this, as

24743 noted in [Section 2.12.3](#) (on page 541). The result of converting a pointer to a function into a

24744 pointer to another data type (except **void \***) is still undefined, however.

24746 Note that compilers conforming to the ISO C standard are required to generate a warning if a

24747 conversion from a **void \*** pointer to a function pointer is attempted as in:

```
24748 fptr = (int (*)(int))dlsym(handle, "my_function");
```

**FUTURE DIRECTIONS**

24749 None.

**SEE ALSO**

24752 [dlclose\(\)](#), [dlerror\(\)](#), [dlopen\(\)](#)

24753 XBD [<dlfcn.h>](#)

**CHANGE HISTORY**

24754 First released in Issue 5.

**Issue 6**

24757 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the

24758 ISO/IEC 9899:1999 standard.

24759 The RTLD\_DEFAULT and RTLD\_NEXT flags are reserved for future use.

24760 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and

24761 adding text to the RATIONALE describing issues related to conversion of pointers to functions

24762 and back again.

**Issue 7**

24764 The *dlsym()* function is moved from the XSI option to the Base.

24765 **NAME**24766 `dprintf` — print formatted output24767 **SYNOPSIS**24768 CX `#include <stdio.h>`24769 `int dprintf(int fildes, const char *restrict format, ...);`24770 **DESCRIPTION**24771 Refer to *fprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**drand48()**24772 **NAME**

24773 **drand48**, **erand48**, **jrand48**, **lcong48**, **lrand48**, **mrnd48**, **nrnd48**, **seed48**, **srand48** — generate  
 24774 uniformly distributed pseudo-random numbers

24775 **SYNOPSIS**

```
24776 XSI #include <stdlib.h>
24777 double drand48(void);
24778 double erand48(unsigned short xsubi[3]);
24779 long jrand48(unsigned short xsubi[3]);
24780 void lcong48(unsigned short param[7]);
24781 long lrand48(void);
24782 long mrnd48(void);
24783 long nrnd48(unsigned short xsubi[3]);
24784 unsigned short *seed48(unsigned short seed16v[3]);
24785 void srand48(long seedval);
```

24786 **DESCRIPTION**

24787 This family of functions shall generate pseudo-random numbers using a linear congruential  
 24788 algorithm and 48-bit integer arithmetic.

24789 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-  
 24790 point values, uniformly distributed over the interval [0,0,1.0).

24791 The *lrand48()* and *nrnd48()* functions shall return non-negative, long integers, uniformly  
 24792 distributed over the interval [0,2<sup>31</sup>).

24793 The *mrnd48()* and *jrand48()* functions shall return signed long integers uniformly distributed  
 24794 over the interval [-2<sup>31</sup>,2<sup>31</sup>).

24795 The *srand48()*, *seed48()*, and *lcong48()* functions are initialization entry points, one of which  
 24796 should be invoked before either *drand48()*, *lrand48()*, or *mrnd48()* is called. (Although it is not  
 24797 recommended practice, constant default initializer values shall be supplied automatically if  
 24798 *drand48()*, *lrand48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)  
 24799 The *erand48()*, *nrnd48()*, and *jrand48()* functions do not require an initialization entry point to  
 24800 be called first.

24801 All the routines work by generating a sequence of 48-bit integer values,  $X_i$ , according to the  
 24802 linear congruential formula:

$$24803 X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

24804 The parameter  $m = 2^{48}$ ; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,  
 24805 the multiplier value  $a$  and the addend value  $c$  are given by:

$$24806 a = 5DEECE66D_{16} = 273673163155_8$$

$$24807 c = B_{16} = 13_8$$

24808 The value returned by any of the *drand48()*, *erand48()*, *jrand48()*, *lrand48()*, *mrnd48()*, or  
 24809 *nrnd48()* functions is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the  
 24810 appropriate number of bits, according to the type of data item to be returned, are copied from  
 24811 the high-order (leftmost) bits of  $X_i$  and transformed into the returned value.

24812 The *drand48()*, *lrand48()*, and *mrnd48()* functions store the last 48-bit  $X_i$  generated in an  
 24813 internal buffer; that is why the application shall ensure that these are initialized prior to being  
 24814 invoked. The *erand48()*, *nrnd48()*, and *jrand48()* functions require the calling program to

24815 provide storage for the successive  $X_i$  values in the array specified as an argument when the  
 24816 functions are invoked. That is why these routines do not have to be initialized; the calling  
 24817 program merely has to place the desired initial value of  $X_i$  into the array and pass it as an  
 24818 argument. By using different arguments, *erand48()*, *nrand48()*, and *rand48()* allow separate  
 24819 modules of a large program to generate several *independent* streams of pseudo-random numbers;  
 24820 that is, the sequence of numbers in each stream shall *not* depend upon how many times the  
 24821 routines are called to generate numbers for the other streams.

24822 The initializer function *srand48()* sets the high-order 32 bits of  $X_i$  to the low-order 32 bits  
 24823 contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330E_{16}$ .

24824 The initializer function *seed48()* sets the value of  $X_i$  to the 48-bit value specified in the argument  
 24825 array. The low-order 16 bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16  
 24826 bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of  $X_i$  are set to the  
 24827 low-order 16 bits of *seed16v*[2]. In addition, the previous value of  $X_i$  is copied into a 48-bit  
 24828 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by  
 24829 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is  
 24830 to be restarted from a given point at some future time—use the pointer to get at and store the  
 24831 last  $X_i$  value, and then use this value to reinitialize via *seed48()* when the program is restarted.

24832 The initializer function *lcong48()* allows the user to specify the initial  $X_i$ , the multiplier value  $a$ ,  
 24833 and the addend value  $c$ . Argument array elements *param*[0-2] specify  $X_i$ , *param*[3-5] specify the  
 24834 multiplier  $a$ , and *param*[6] specifies the 16-bit addend  $c$ . After *lcong48()* is called, a subsequent  
 24835 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values,  $a$  and  
 24836  $c$ , specified above.

24837 The *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be thread-safe.

#### 24838 RETURN VALUE

24839 As described in the DESCRIPTION above.

#### 24840 ERRORS

24841 No errors are defined.

#### 24842 EXAMPLES

24843 None.

#### 24844 APPLICATION USAGE

24845 None.

#### 24846 RATIONALE

24847 None.

#### 24848 FUTURE DIRECTIONS

24849 None.

#### 24850 SEE ALSO

24851 [rand\(\)](#)

24852 XBD [<stdlib.h>](#)

#### 24853 CHANGE HISTORY

24854 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 24855 Issue 5

24856 A note indicating that the *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant is  
 24857 added to the DESCRIPTION.

## drand48()

24858 **Issue 6**

24859 The normative text is updated to avoid use of the term “must” for application requirements.

24860 **Issue 7**

24861 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

24862 **NAME**

24863 dup, dup2 — duplicate an open file descriptor

24864 **SYNOPSIS**

```
24865 #include <unistd.h>
24866 int dup(int fildes);
24867 int dup2(int fildes, int fildes2);
```

24868 **DESCRIPTION**

24869 The *dup()* function provides an alternative interface to the service provided by *fcntl()* using the  
 24870 F\_DUPFD command. The call *dup(fildes)* shall be equivalent to:

```
24871 fcntl(fildes, F_DUPFD, 0);
```

24872 The *dup2()* function shall cause the file descriptor *fildes2* to refer to the same open file  
 24873 description as the file descriptor *fildes* and to share any locks, and shall return *fildes2*. If *fildes2* is  
 24874 already a valid open file descriptor, it shall be closed first, unless *fildes* is equal to *fildes2* in which  
 24875 case *dup2()* shall return *fildes2* without closing it. If the close operation fails to close *fildes2*,  
 24876 *dup2()* shall return  $-1$  without changing the open file description to which *fildes2* refers. If *fildes*  
 24877 is not a valid file descriptor, *dup2()* shall return  $-1$  and shall not close *fildes2*. If *fildes2* is less than  
 24878 0 or greater than or equal to {OPEN\_MAX}, *dup2()* shall return  $-1$  with *errno* set to [EBADF].

24879 Upon successful completion, if *fildes* is not equal to *fildes2*, the FD\_CLOEXEC flag associated  
 24880 with *fildes2* shall be cleared. If *fildes* is equal to *fildes2*, the FD\_CLOEXEC flag associated with  
 24881 *fildes2* shall not be changed.

24882 TYM If *fildes* refers to a typed memory object, the result of the *dup2()* function is unspecified.

24883 **RETURN VALUE**

24884 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;  
 24885 otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

24886 **ERRORS**

24887 The *dup()* function shall fail if:

24888 [EBADF] The *fildes* argument is not a valid open file descriptor.

24889 [EMFILE] All file descriptors available to the process are currently open.

24890 The *dup2()* function shall fail if:

24891 [EBADF] The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is  
 24892 negative or greater than or equal to {OPEN\_MAX}.

24893 [EINTR] The *dup2()* function was interrupted by a signal.

24894 The *dup2()* function may fail if:

24895 [EIO] An I/O error occurred while attempting to close *fildes2*.

24896 **EXAMPLES**24897 **Redirecting Standard Output to a File**

24898 The following example closes standard output for the current processes, re-assigns standard  
 24899 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
24900 #include <unistd.h>
24901 ...
24902 int pfid;
24903 ...
```

**dup()**

```

24904     close(1);
24905     dup(pfd);
24906     close(pfd);
24907     ...

```

### 24908 **Redirecting Error Messages**

24909 The following example redirects messages from *stderr* to *stdout*.

```

24910     #include <unistd.h>
24911     ...
24912     dup2(1, 2);
24913     ...

```

### 24914 **APPLICATION USAGE**

24915 None.

### 24916 **RATIONALE**

24917 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*  
 24918 function. They have been included in this volume of POSIX.1-2008 primarily for historical  
 24919 reasons, since many existing applications use them.

24920 The *dup2()* function is not marked obsolescent because it presents a type-safe version of  
 24921 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

24922 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

24923 In the description of [EBADF], the case of *fildev* being out of range is covered by the given case of  
 24924 *fildev* not being valid. The descriptions for *fildev* and *fildev2* are different because the only kind of  
 24925 invalidity that is relevant for *fildev2* is whether it is out of range; that is, it does not matter  
 24926 whether *fildev2* refers to an open file when the *dup2()* call is made.

### 24927 **FUTURE DIRECTIONS**

24928 None.

### 24929 **SEE ALSO**

24930 *close()*, *fcntl()*, *open()*

24931 XBD <[unistd.h](#)>

### 24932 **CHANGE HISTORY**

24933 First released in Issue 1. Derived from Issue 1 of the SVID.

### 24934 **Issue 7**

24935 SD5-XSH-ERN-187 is applied.

24936 **NAME**

24937 duplocale — duplicate a locale object

24938 **SYNOPSIS**

```
24939 CX #include <locale.h>
24940 locale_t duplocale(locale_t locobj);
```

24941 **DESCRIPTION**

24942 The *duplocale()* function shall create a duplicate copy of the locale object referenced by the *locobj*  
 24943 argument.

24944 **RETURN VALUE**

24945 Upon successful completion, the *duplocale()* function shall return a handle for a new locale  
 24946 object. Otherwise, *duplocale()* shall return **(locale\_t)0** and set *errno* to indicate the error.

24947 **ERRORS**

24948 The *duplocale()* function shall fail if:

24949 [ENOMEM] There is not enough memory available to create the locale object or load the  
 24950 locale data.

24951 The *duplocale()* function may fail if:

24952 [EINVAL] *locobj* is not a handle for a locale object.

24953 **EXAMPLES**24954 **Constructing an Altered Version of an Existing Locale Object**

24955 The following example shows a code fragment to create a slightly altered version of an existing  
 24956 locale object. The function takes a locale object and a locale name and it replaces the *LC\_TIME*  
 24957 category data in the locale object with that from the named locale.

```
24958 #include <locale.h>
24959 ...
24960 locale_t
24961 with_changed_lc_time (locale_t obj, const char *name)
24962 {
24963     locale_t retval = duplocale (obj);
24964     if (retval != (locale_t) 0)
24965     {
24966         locale_t changed = newlocale (LC_TIME_MASK, name, retval);
24967         if (changed == (locale_t) 0)
24968             /* An error occurred. Free all allocated resources. */
24969             freelocale (retval);
24970         retval = changed;
24971     }
24972     return retval; }
24973 }
```

24974 **APPLICATION USAGE**

24975 The use of the *duplocale()* function is recommended for situations where a locale object is being  
 24976 used in multiple places, and it is possible that the lifetime of the locale object might end before  
 24977 all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is  
 24978 needed. This can be achieved by a call to *newlocale()* following the *duplocale()* call.

**duplocale()**

24979 As with the *newlocale()* function, handles for locale objects created by the *duplocale()* function  
24980 should be released by a corresponding call to *freelocale()*.

24981 **RATIONALE**

24982 None.

24983 **FUTURE DIRECTIONS**

24984 None.

24985 **SEE ALSO**24986 *freelocale()*, *newlocale()*, *uselocale()*

24987 XBD &lt;locale.h&gt;

24988 **CHANGE HISTORY**

24989 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

24990 **NAME**24991 encrypt — encoding function (**CRYPT**)24992 **SYNOPSIS**

```
24993 XSI #include <unistd.h>
24994 void encrypt(char block[64], int edflag);
```

24995 **DESCRIPTION**

24996 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.  
 24997 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

24998 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes  
 24999 with values of 0 and 1. The array is modified in place to a similar array using the key set by  
 25000 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see  
 25001 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to  
 25002 [ENOSYS].

25003 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing  
 25004 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on  
 25005 return, an error has occurred.

25006 The *encrypt()* function need not be thread-safe.

25007 **RETURN VALUE**

25008 The *encrypt()* function shall not return a value.

25009 **ERRORS**

25010 The *encrypt()* function shall fail if:

25011 [ENOSYS] The functionality is not supported on this implementation.

25012 **EXAMPLES**

25013 None.

25014 **APPLICATION USAGE**

25015 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.

25016 In some environments, decoding might not be implemented. This is related to some Government  
 25017 restrictions on encryption and decryption routines. Historical practice has been to ship a  
 25018 different version of the encryption library without the decryption feature in the routines  
 25019 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

25020 **RATIONALE**

25021 None.

25022 **FUTURE DIRECTIONS**

25023 None.

25024 **SEE ALSO**

25025 *crypt()*, *setkey()*

25026 XBD <unistd.h>

25027 **CHANGE HISTORY**

25028 First released in Issue 1. Derived from Issue 1 of the SVID.

**encrypt()**

- 25029 **Issue 5**  
25030 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 25031 **Issue 6**  
25032 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 25033 **Issue 7**  
25034 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

25035 **NAME**

25036 endgrent, getgrent, setgrent — group database entry functions

25037 **SYNOPSIS**

```

25038 xSI      #include <grp.h>
25039
25039      void endgrent(void);
25040      struct group *getgrent(void);
25041      void setgrent(void);

```

25042 **DESCRIPTION**

25043 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an  
 25044 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**  
 25045 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a  
 25046 **group** structure containing the next group structure in the group database, so successive calls  
 25047 may be used to search the entire database.

25048 An implementation that provides extended security controls may impose further  
 25049 implementation-defined restrictions on accessing the group database. In particular, the system  
 25050 may deny the existence of some or all of the group database entries associated with groups other  
 25051 than those groups associated with the caller and may omit users other than the caller from the  
 25052 list of members of groups in database entries that are returned.

25053 The *setgrent()* function shall rewind the group database to allow repeated searches.

25054 The *endgrent()* function may be called to close the group database when processing is complete.

25055 These functions need not be thread-safe.

25056 **RETURN VALUE**

25057 When first called, *getgrent()* shall return a pointer to the first group structure in the group  
 25058 database. Upon subsequent calls it shall return the next group structure in the group database.  
 25059 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set  
 25060 to indicate the error.

25061 The return value may point to a static area which is overwritten by a subsequent call to  
 25062 *getgrgid()*, *getgrnam()*, or *getgrent()*.

25063 **ERRORS**

25064 The *getgrent()* function may fail if:

- |       |          |                                                                        |
|-------|----------|------------------------------------------------------------------------|
| 25065 | [EINTR]  | A signal was caught during the operation.                              |
| 25066 | [EIO]    | An I/O error has occurred.                                             |
| 25067 | [EMFILE] | All file descriptors available to the process are currently open.      |
| 25068 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

25069 **EXAMPLES**

25070 None.

25071 **APPLICATION USAGE**

25072 These functions are provided due to their historical usage. Applications should avoid  
 25073 dependencies on fields in the group database, whether the database is a single file, or where in  
 25074 the file system name space the database resides. Applications should use *getgrnam()* and  
 25075 *getgrgid()* whenever possible because it avoids these dependencies.

**endgrent()**

System Interfaces

25076 **RATIONALE**

25077 None.

25078 **FUTURE DIRECTIONS**

25079 None.

25080 **SEE ALSO**25081 *endpwent()*, *getgrgid()*, *getgrnam()*, *getlogin()*

25082 XBD &lt;grp.h&gt;

25083 **CHANGE HISTORY**

25084 First released in Issue 4, Version 2.

25085 **Issue 5**

25086 Moved from X/OPEN UNIX extension to BASE.

25087 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
25088 VALUE section.

25089 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25090 **Issue 6**

25091 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

25092 **Issue 7**

25093 Austin Group Interpretation 1003.1-2001 #156 is applied.

25094 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

25095 **NAME**

25096 endhostent, gethostent, sethostent — network host database functions

25097 **SYNOPSIS**

```
25098 #include <netdb.h>
25099 void endhostent(void);
25100 struct hostent *gethostent(void);
25101 void sethostent(int stayopen);
```

25102 **DESCRIPTION**

25103 These functions shall retrieve information about hosts. This information is considered to be  
 25104 stored in a database that can be accessed sequentially or randomly. The implementation of this  
 25105 database is unspecified.

25106 **Note:** In many cases this database is implemented by the Domain Name System, as documented in  
 25107 RFC 1034, RFC 1035, and RFC 1886.

25108 The *sethostent()* function shall open a connection to the database and set the next entry for  
 25109 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection  
 25110 shall not be closed by a call to *gethostent()*, and the implementation may maintain an open file  
 25111 descriptor.

25112 The *gethostent()* function shall read the next entry in the database, opening and closing a  
 25113 connection to the database as necessary.

25114 Entries shall be returned in **hostent** structures.

25115 The *endhostent()* function shall close the connection to the database, releasing any open file  
 25116 descriptor.

25117 These functions need not be thread-safe.

25118 **RETURN VALUE**

25119 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**  
 25120 structure if the requested entry was found, and a null pointer if the end of the database was  
 25121 reached or the requested entry was not found.

25122 **ERRORS**

25123 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.

25124 **EXAMPLES**

25125 None.

25126 **APPLICATION USAGE**

25127 The *gethostent()* function may return pointers to static data, which may be overwritten by  
 25128 subsequent calls to any of these functions.

25129 **RATIONALE**

25130 None.

25131 **FUTURE DIRECTIONS**

25132 None.

25133 **SEE ALSO**

25134 *endservent()*

25135 XBD [<netdb.h>](#)

**endhostent()**

25136 **CHANGE HISTORY**

25137 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25138 **Issue 7**

25139 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

25140 **NAME**

25141 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

25142 **SYNOPSIS**

```

25143     #include <netdb.h>

25144     void endnetent(void);
25145     struct netent *getnetbyaddr(uint32_t net, int type);
25146     struct netent *getnetbyname(const char *name);
25147     struct netent *getnetent(void);
25148     void setnetent(int stayopen);

```

25149 **DESCRIPTION**

25150 These functions shall retrieve information about networks. This information is considered to be  
 25151 stored in a database that can be accessed sequentially or randomly. The implementation of this  
 25152 database is unspecified.

25153 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-  
 25154 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either  
 25155 directly, or indirectly through one of the other *getnet\**(*)* functions), and the implementation may  
 25156 maintain an open file descriptor to the database.

25157 The *getnetent()* function shall read the next entry of the database, opening and closing a  
 25158 connection to the database as necessary.

25159 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry  
 25160 for which the address family specified by *type* matches the *n\_addrtype* member and the network  
 25161 number *net* matches the *n\_net* member, opening and closing a connection to the database as  
 25162 necessary. The *net* argument shall be the network number in host byte order.

25163 The *getnetbyname()* function shall search the database from the beginning and find the first entry  
 25164 for which the network name specified by *name* matches the *n\_name* member, opening and  
 25165 closing a connection to the database as necessary.

25166 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a  
 25167 **netent** structure, the members of which shall contain the fields of an entry in the network  
 25168 database.

25169 The *endnetent()* function shall close the database, releasing any open file descriptor.

25170 These functions need not be thread-safe.

25171 **RETURN VALUE**

25172 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a  
 25173 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the  
 25174 database was reached or the requested entry was not found. Otherwise, a null pointer shall be  
 25175 returned.

25176 **ERRORS**

25177 No errors are defined.

**endnetent()**

System Interfaces

25178 **EXAMPLES**

25179 None.

25180 **APPLICATION USAGE**

25181 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions may return pointers to static data,  
25182 which may be overwritten by subsequent calls to any of these functions.

25183 **RATIONALE**

25184 None.

25185 **FUTURE DIRECTIONS**

25186 None.

25187 **SEE ALSO**25188 XBD [<netdb.h>](#)25189 **CHANGE HISTORY**

25190 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25191 **Issue 7**

25192 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

25193 **NAME**

25194 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol  
25195 database functions

25196 **SYNOPSIS**

```
25197 #include <netdb.h>
25198 void endprotoent(void);
25199 struct protoent *getprotobyname(const char *name);
25200 struct protoent *getprotobynumber(int proto);
25201 struct protoent *getprotoent(void);
25202 void setprotoent(int stayopen);
```

25203 **DESCRIPTION**

25204 These functions shall retrieve information about protocols. This information is considered to be  
25205 stored in a database that can be accessed sequentially or randomly. The implementation of this  
25206 database is unspecified.

25207 The *setprotoent()* function shall open a connection to the database, and set the next entry to the  
25208 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database  
25209 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the  
25210 other *getproto\**(*)* functions), and the implementation may maintain an open file descriptor for  
25211 the database.

25212 The *getprotobyname()* function shall search the database from the beginning and find the first  
25213 entry for which the protocol name specified by *name* matches the *p\_name* member, opening and  
25214 closing a connection to the database as necessary.

25215 The *getprotobynumber()* function shall search the database from the beginning and find the first  
25216 entry for which the protocol number specified by *proto* matches the *p\_proto* member, opening  
25217 and closing a connection to the database as necessary.

25218 The *getprotoent()* function shall read the next entry of the database, opening and closing a  
25219 connection to the database as necessary.

25220 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer  
25221 to a **protoent** structure, the members of which shall contain the fields of an entry in the network  
25222 protocol database.

25223 The *endprotoent()* function shall close the connection to the database, releasing any open file  
25224 descriptor.

25225 These functions need not be thread-safe.

25226 **RETURN VALUE**

25227 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a  
25228 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of  
25229 the database was reached or the requested entry was not found. Otherwise, a null pointer is  
25230 returned.

25231 **ERRORS**

25232 No errors are defined.

**endprotoent()**

System Interfaces

25233 **EXAMPLES**

25234 None.

25235 **APPLICATION USAGE**25236 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to  
25237 static data, which may be overwritten by subsequent calls to any of these functions.25238 **RATIONALE**

25239 None.

25240 **FUTURE DIRECTIONS**

25241 None.

25242 **SEE ALSO**25243 XBD [<netdb.h>](#)25244 **CHANGE HISTORY**

25245 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25246 **Issue 7**

25247 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

25248 **NAME**

25249 endpwent, getpwent, setpwent — user database functions

25250 **SYNOPSIS**

```

25251 XSI #include <pwd.h>
25252 void endpwent(void);
25253 struct passwd *getpwent(void);
25254 void setpwent(void);

```

25255 **DESCRIPTION**

25256 These functions shall retrieve information about users.

25257 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of  
 25258 an entry in the user database. Each entry in the user database contains a **passwd** structure. When  
 25259 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in  
 25260 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next  
 25261 entry in the user database. Successive calls can be used to search the entire user database.

25262 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

25263 An implementation that provides extended security controls may impose further  
 25264 implementation-defined restrictions on accessing the user database. In particular, the system  
 25265 may deny the existence of some or all of the user database entries associated with users other  
 25266 than the caller.

25267 The *setpwent()* function effectively rewinds the user database to allow repeated searches.25268 The *endpwent()* function may be called to close the user database when processing is complete.

25269 These functions need not be thread-safe.

25270 **RETURN VALUE**25271 The *getpwent()* function shall return a null pointer on end-of-file or error.25272 **ERRORS**

25273 These functions may fail if:

25274 [EIO] An I/O error has occurred.

25275 In addition, *getpwent()* and *setpwent()* may fail if:

25276 [EMFILE] All file descriptors available to the process are currently open.

25277 [ENFILE] The maximum allowable number of files is currently open in the system.

25278 The return value may point to a static area which is overwritten by a subsequent call to  
 25279 *getpwnid()*, *getpwnam()*, or *getpwent()*.

**endpwent()**25280 **EXAMPLES**25281 **Searching the User Database**

25282 The following example uses the *getpwent()* function to get successive entries in the user  
 25283 database, returning a pointer to a **passwd** structure that contains information about each user.  
 25284 The call to *endpwent()* closes the user database and cleans up.

```

25285 #include <pwd.h>
25286 #include <stdio.h>

25287 void printname(uid_t uid)
25288 {
25289     struct passwd *pwd;

25290     setpwent();
25291     while((pwd = getpwent()) != NULL) {
25292         if (pwd->pw_uid == uid) {
25293             printf("name=%s\n", pwd->pw_name);
25294             break;
25295         }
25296     }
25297     endpwent();
25298 }

```

25299 **APPLICATION USAGE**

25300 These functions are provided due to their historical usage. Applications should avoid  
 25301 dependencies on fields in the password database, whether the database is a single file, or where  
 25302 in the file system name space the database resides. Applications should use *getpwuid()*  
 25303 whenever possible because it avoids these dependencies.

25304 **RATIONALE**

25305 None.

25306 **FUTURE DIRECTIONS**

25307 None.

25308 **SEE ALSO**

25309 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*

25310 XBD **<pwd.h>**

25311 **CHANGE HISTORY**

25312 First released in Issue 4, Version 2.

25313 **Issue 5**

25314 Moved from X/OPEN UNIX extension to BASE.

25315 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 25316 VALUE section.

25317 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25318 **Issue 6**

25319 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

- 25320 **Issue 7**
- 25321 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 25322 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 25323 The EXAMPLES section is revised.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**endservent()**25324 **NAME**

25325 endservent, getservbyname, getservbyport, getservent, setservent — network services database  
 25326 functions

25327 **SYNOPSIS**

```
25328 #include <netdb.h>
25329
25329 void endservent(void);
25330 struct servent *getservbyname(const char *name, const char *proto);
25331 struct servent *getservbyport(int port, const char *proto);
25332 struct servent *getservent(void);
25333 void setservent(int stayopen);
```

25334 **DESCRIPTION**

25335 These functions shall retrieve information about network services. This information is  
 25336 considered to be stored in a database that can be accessed sequentially or randomly. The  
 25337 implementation of this database is unspecified.

25338 The *setservent()* function shall open a connection to the database, and set the next entry to the  
 25339 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each  
 25340 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv\**(  
 25341 functions), and the implementation may maintain an open file descriptor for the database.

25342 The *getservent()* function shall read the next entry of the database, opening and closing a  
 25343 connection to the database as necessary.

25344 The *getservbyname()* function shall search the database from the beginning and find the first  
 25345 entry for which the service name specified by *name* matches the *s\_name* member and the protocol  
 25346 name specified by *proto* matches the *s\_proto* member, opening and closing a connection to the  
 25347 database as necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be  
 25348 matched.

25349 The *getservbyport()* function shall search the database from the beginning and find the first entry  
 25350 for which the port specified by *port* matches the *s\_port* member and the protocol name specified  
 25351 by *proto* matches the *s\_proto* member, opening and closing a connection to the database as  
 25352 necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be matched. The *port*  
 25353 argument shall be a value obtained by converting a **uint16\_t** in network byte order to **int**.

25354 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a  
 25355 **servent** structure, the members of which shall contain the fields of an entry in the network  
 25356 services database.

25357 The *endservent()* function shall close the database, releasing any open file descriptor.

25358 These functions need not be thread-safe.

25359 **RETURN VALUE**

25360 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to  
 25361 a **servent** structure if the requested entry was found, and a null pointer if the end of the database  
 25362 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

25363 **ERRORS**

25364 No errors are defined.

25365 **EXAMPLES**

25366 None.

25367 **APPLICATION USAGE**25368 The *port* argument of *getserbyport()* need not be compatible with the port values of all address  
25369 families.25370 The *getserbyname()*, *getserbyport()*, and *getservent()* functions may return pointers to static  
25371 data, which may be overwritten by subsequent calls to any of these functions.25372 **RATIONALE**

25373 None.

25374 **FUTURE DIRECTIONS**

25375 None.

25376 **SEE ALSO**25377 *endhostent()*, *endprotoent()*, *htonl()*, *inet\_addr()*25378 XBD <[netdb.h](#)>25379 **CHANGE HISTORY**

25380 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25381 **Issue 7**

25382 Austin Group Interpretation 1003.1-2001 #156 is applied.

25383 SD5-XBD-ERN-14 is applied.

**endutxent()**25384 **NAME**

25385 **endutxent**, **getutxent**, **getutxid**, **getutxline**, **pututxline**, **setutxent** — user accounting database  
 25386 functions

25387 **SYNOPSIS**

```
25388 xSI #include <utmpx.h>
25389 void endutxent(void);
25390 struct utmpx *getutxent(void);
25391 struct utmpx *getutxid(const struct utmpx *id);
25392 struct utmpx *getutxline(const struct utmpx *line);
25393 struct utmpx *pututxline(const struct utmpx *utmpx);
25394 void setutxent(void);
```

25395 **DESCRIPTION**

25396 These functions shall provide access to the user accounting database.

25397 The *getutxent()* function shall read the next entry from the user accounting database. If the  
 25398 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

25399 The *getutxid()* function shall search forward from the current point in the database. If the  
 25400 *ut\_type* value of the **utmpx** structure pointed to by *id* is *BOOT\_TIME*, *OLD\_TIME*, or  
 25401 *NEW\_TIME*, then it shall stop when it finds an entry with a matching *ut\_type* value. If the  
 25402 *ut\_type* value is *INIT\_PROCESS*, *LOGIN\_PROCESS*, *USER\_PROCESS*, or *DEAD\_PROCESS*,  
 25403 then it shall stop when it finds an entry whose type is one of these four and whose *ut\_id* member  
 25404 matches the *ut\_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is  
 25405 reached without a match, *getutxid()* shall fail.

25406 The *getutxline()* function shall search forward from the current point in the database until it  
 25407 finds an entry of the type *LOGIN\_PROCESS* or *USER\_PROCESS* which also has a *ut\_line* value  
 25408 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached  
 25409 without a match, *getutxline()* shall fail.

25410 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to  
 25411 search for multiple occurrences, the application shall zero out the static data after each success,  
 25412 or *getutxline()* may return a pointer to the same **utmpx** structure.

25413 There is one exception to the rule about clearing the structure before further reads are done. The  
 25414 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user  
 25415 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or  
 25416 *getutxline()*, if the application has modified this structure and passed the pointer back to  
 25417 *pututxline()*.

25418 For all entries that match a request, the *ut\_type* member indicates the type of the entry. Other  
 25419 members of the entry shall contain meaningful data based on the value of the *ut\_type* member as  
 25420 follows:

| 25421 | ut_type Member | Other Members with Meaningful Data                                                             |
|-------|----------------|------------------------------------------------------------------------------------------------|
| 25422 | EMPTY          | No others                                                                                      |
| 25423 | BOOT_TIME      | <i>ut_tv</i>                                                                                   |
| 25424 | OLD_TIME       | <i>ut_tv</i>                                                                                   |
| 25425 | NEW_TIME       | <i>ut_tv</i>                                                                                   |
| 25426 | USER_PROCESS   | <i>ut_id, ut_user</i> (login name of the user), <i>ut_line, ut_pid, ut_tv</i>                  |
| 25427 | INIT_PROCESS   | <i>ut_id, ut_pid, ut_tv</i>                                                                    |
| 25428 | LOGIN_PROCESS  | <i>ut_id, ut_user</i> (implementation-defined name of the login process), <i>ut_pid, ut_tv</i> |
| 25429 |                |                                                                                                |
| 25430 | DEAD_PROCESS   | <i>ut_id, ut_pid, ut_tv</i>                                                                    |

25431 An implementation that provides extended security controls may impose implementation-  
 25432 defined restrictions on accessing the user accounting database. In particular, the system may  
 25433 deny the existence of some or all of the user accounting database entries associated with users  
 25434 other than the caller.

25435 If the process has appropriate privileges, the *pututxline()* function shall write out the structure  
 25436 into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the  
 25437 request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be  
 25438 made at the end of the user accounting database.

25439 The *endutxent()* function shall close the user accounting database.

25440 The *setutxent()* function shall reset the input to the beginning of the database. This should be  
 25441 done before each search for a new entry if it is desired that the entire database be examined.

25442 These functions need not be thread-safe.

#### 25443 RETURN VALUE

25444 Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a  
 25445 **utmpx** structure containing a copy of the requested entry in the user accounting database.  
 25446 Otherwise, a null pointer shall be returned.

25447 The return value may point to a static area which is overwritten by a subsequent call to  
 25448 *getutxid()* or *getutxline()*.

25449 Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a  
 25450 copy of the entry added to the user accounting database. Otherwise, a null pointer shall be  
 25451 returned.

25452 The *endutxent()* and *setutxent()* functions shall not return a value.

#### 25453 ERRORS

25454 No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()*  
 25455 functions.

25456 The *pututxline()* function may fail if:

25457 [EPERM] The process does not have appropriate privileges.

**endutxent()**

System Interfaces

25458 **EXAMPLES**

25459 None.

25460 **APPLICATION USAGE**25461 The sizes of the arrays in the structure can be found using the *sizeof* operator.25462 **RATIONALE**

25463 None.

25464 **FUTURE DIRECTIONS**

25465 None.

25466 **SEE ALSO**25467 XBD [<utmpx.h>](#)25468 **CHANGE HISTORY**

25469 First released in Issue 4, Version 2.

25470 **Issue 5**

25471 Moved from X/OPEN UNIX extension to BASE.

25472 Normative text previously in the APPLICATION USAGE section is moved to the  
25473 DESCRIPTION.

25474 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25475 **Issue 6**

25476 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

25477 **Issue 7**

25478 Austin Group Interpretation 1003.1-2001 #156 is applied.

*System Interfaces***environ**25479 **NAME**

25480 environ — array of character pointers to the environment strings

25481 **SYNOPSIS**

25482 extern char \*\*environ;

25483 **DESCRIPTION**25484 Refer to *exec* and XBD [Chapter 8](#) (on page 173).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**erand48()***System Interfaces*25485 **NAME**25486            **erand48** — generate uniformly distributed pseudo-random numbers25487 **SYNOPSIS**

```
25488 XSI        #include <stdlib.h>  
25489            double erand48(unsigned short xsubi[3]);
```

25490 **DESCRIPTION**25491            Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

25492 **NAME**

25493 erf, erff, erfl — error functions

25494 **SYNOPSIS**

```
25495 #include <math.h>
25496 double erf(double x);
25497 float erff(float x);
25498 long double erfl(long double x);
```

25499 **DESCRIPTION**

25500 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 25501 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25502 volume of POSIX.1-2008 defers to the ISO C standard.

25503 These functions shall compute the error function of their argument  $x$ , defined as:

$$25504 \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

25505 An application wishing to check for error situations should set *errno* to zero and call  
 25506 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 25507 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 25508 zero, an error has occurred.

25509 **RETURN VALUE**

25510 Upon successful completion, these functions shall return the value of the error function.

25511 MX If  $x$  is NaN, a NaN shall be returned.

25512 If  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.

25513 If  $x$  is  $\pm \text{Inf}$ ,  $\pm 1$  shall be returned.

25514 If  $x$  is subnormal, a range error may occur, and  $2 * x / \text{sqrt}(\pi)$  should be returned.

25515 **ERRORS**

25516 These functions may fail if:

25517 MX **Range Error** The result underflows.

25518 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 25519 then *errno* shall be set to [ERANGE]. If the integer expression  
 25520 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 25521 floating-point exception shall be raised.

25522 **EXAMPLES**25523 **Computing the Probability for a Normal Variate**

25524 This example shows how to use *erf()* to compute the probability that a normal variate assumes a  
 25525 value in the range  $[x_1, x_2]$  with  $x_1 \leq x_2$ .

25526 This example uses the constant M\_SQRT1\_2 which is part of the XSI option.

```
25527 #include <math.h>
25528 double
25529 Phi(const double x1, const double x2)
25530 {
25531     return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
```

**erf()**

25532            }

25533 **APPLICATION USAGE**

25534            Underflow occurs when  $|x| < \text{DBL\_MIN} * (\text{sqrt}(\pi)/2)$ .

25535            On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
25536 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

25537 **RATIONALE**

25538            None.

25539 **FUTURE DIRECTIONS**

25540            None.

25541 **SEE ALSO**

25542            *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

25543            XBD Section 4.19 (on page 116), **<math.h>**

25544 **CHANGE HISTORY**

25545            First released in Issue 1. Derived from Issue 1 of the SVID.

25546 **Issue 5**

25547            The DESCRIPTION is updated to indicate how an application should check for an error. This  
25548 text was previously published in the APPLICATION USAGE section.

25549 **Issue 6**

25550            The *erf()* function is no longer marked as an extension.

25551            The *erfc()* function is split out onto its own reference page.

25552            The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

25553            The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
25554 revised to align with the ISO/IEC 9899:1999 standard.

25555            IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
25556 marked.

25557            IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the  
25558 EXAMPLES section.

25559 **NAME**25560 `erfc`, `erfcf`, `erfcl` — complementary error functions25561 **SYNOPSIS**

```
25562 #include <math.h>
25563 double erfc(double x);
25564 float erfcf(float x);
25565 long double erfcl(long double x);
```

25566 **DESCRIPTION**

25567 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 25568 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25569 volume of POSIX.1-2008 defers to the ISO C standard.

25570 These functions shall compute the complementary error function  $1.0 - \text{erf}(x)$ .

25571 An application wishing to check for error situations should set `errno` to zero and call  
 25572 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 25573 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 25574 zero, an error has occurred.

25575 **RETURN VALUE**

25576 Upon successful completion, these functions shall return the value of the complementary error  
 25577 function.

25578 If the correct value would cause underflow and is not representable, a range error may occur  
 25579 **MX** and either 0.0 (if representable), or an implementation-defined value shall be returned.

25580 **MX** If  $x$  is NaN, a NaN shall be returned.

25581 If  $x$  is  $\pm 0$ , +1 shall be returned.

25582 If  $x$  is  $-\text{Inf}$ , +2 shall be returned.

25583 If  $x$  is  $+\text{Inf}$ , +0 shall be returned.

25584 If the correct value would cause underflow and is representable, a range error may occur and  
 25585 the correct value shall be returned.

25586 **ERRORS**

25587 These functions may fail if:

25588 **Range Error** The result underflows.

25589 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
 25590 then `errno` shall be set to `[ERANGE]`. If the integer expression  
 25591 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the underflow  
 25592 floating-point exception shall be raised.

25593 **EXAMPLES**

25594 None.

25595 **APPLICATION USAGE**

25596 The `erfc()` function is provided because of the extreme loss of relative accuracy if `erf(x)` is called  
 25597 for large  $x$  and the result subtracted from 1.0.

25598 Note for IEEE Std 754-1985 **double**,  $26.55 < x$  implies `erfc(x)` has underflowed.

25599 On error, the expressions (`math_errhandling` & `MATH_ERRNO`) and (`math_errhandling` &  
 25600 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**erfc()**25601 **RATIONALE**

25602 None.

25603 **FUTURE DIRECTIONS**

25604 None.

25605 **SEE ALSO**25606 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

25607 XBD Section 4.19 (on page 116), &lt;math.h&gt;

25608 **CHANGE HISTORY**

25609 First released in Issue 1. Derived from Issue 1 of the SVID.

25610 **Issue 5**25611 The DESCRIPTION is updated to indicate how an application should check for an error. This  
25612 text was previously published in the APPLICATION USAGE section.25613 **Issue 6**25614 The *erfc()* function is no longer marked as an extension.25615 These functions are split out from the *erf()* reference page.25616 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
25617 revised to align with the ISO/IEC 9899:1999 standard.25618 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
25619 marked.

*System Interfaces***erff()**25620 **NAME**25621 `erff, erfl` — error functions25622 **SYNOPSIS**25623 `#include <math.h>`25624 `float erff(float x);`25625 `long double erfl(long double x);`25626 **DESCRIPTION**25627 Refer to *erf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**errno**25628 **NAME**25629 `errno` — error return value25630 **SYNOPSIS**25631 `#include <errno.h>`25632 **DESCRIPTION**25633 The lvalue *errno* is used by many functions to return error values.

25634 Many functions provide an error number in *errno*, which has type **int** and is defined in  
 25635 **<errno.h>**. The value of *errno* shall be defined only after a call to a function for which it is  
 25636 explicitly stated to be set and until it is changed by the next function call or if the application  
 25637 assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by  
 25638 a function's return value. Applications shall obtain the definition of *errno* by the inclusion of  
 25639 **<errno.h>**. No function in this volume of POSIX.1-2008 shall set *errno* to 0. The setting of *errno*  
 25640 after a successful call to a function is unspecified unless the description of that function specifies  
 25641 that *errno* shall not be modified.

25642 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a  
 25643 macro definition is suppressed in order to access an actual object, or a program defines an  
 25644 identifier with the name *errno*, the behavior is undefined.

25645 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant  
 25646 pages.

25647 **RETURN VALUE**

25648 None.

25649 **ERRORS**

25650 None.

25651 **EXAMPLES**

25652 None.

25653 **APPLICATION USAGE**

25654 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard  
 25655 in that they required *errno* to be defined as an external variable, whereas the ISO C standard  
 25656 required only that *errno* be defined as a modifiable lvalue with type **int**.

25657 An application that needs to examine the value of *errno* to determine the error should set it to 0  
 25658 before a function call, then inspect it before a subsequent function call.

25659 **RATIONALE**

25660 None.

25661 **FUTURE DIRECTIONS**

25662 None.

25663 **SEE ALSO**25664 [Section 2.3](#)25665 XBD **<errno.h>**25666 **CHANGE HISTORY**

25667 First released in Issue 1. Derived from Issue 1 of the SVID.

25668 **Issue 5**

25669 The following sentence is deleted from the DESCRIPTION: "The value of *errno* is 0 at program  
 25670 start-up, but is never set to 0 by any XSI function".

25671 The DESCRIPTION also no longer states that conforming implementations may support the  
25672 declaration:

25673 `extern int errno;`

25674 **Issue 6**

25675 Obsolescent text regarding defining *errno* as:

25676 `extern int errno`

25677 is removed.

25678 Text regarding no function setting *errno* to zero to indicate an error is changed to no function  
25679 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

25680 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the  
25681 DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified  
25682 unless the description of the function requires that it will not be modified.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 25683 NAME

25684 environ, execl, execl, execlp, execv, execve, execvp, fexecve — execute a file

## 25685 SYNOPSIS

```

25686 #include <unistd.h>
25687 extern char **environ;
25688 int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
25689 int execlp(const char *path, const char *arg0, ... /*,
25690 (char *)0, char *const envp[] */);
25691 int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
25692 int execv(const char *path, char *const argv[]);
25693 int execve(const char *path, char *const argv[], char *const envp[]);
25694 int execvp(const char *file, char *const argv[]);
25695 int fexecve(int fd, char *const argv[], char *const envp[]);

```

## 25696 DESCRIPTION

25697 The *exec* family of functions shall replace the current process image with a new process image.  
 25698 The new image shall be constructed from a regular, executable file called the *new process image*  
 25699 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid  
 25700 by the new process image.

25701 The *fexecve()* function shall be equivalent to the *execve()* function except that the file to be  
 25702 executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is  
 25703 ignored.

25704 When a C-language program is executed as a result of a call to one of the *exec* family of  
 25705 functions, it shall be entered as a C-language function call as follows:

```
25706 int main (int argc, char *argv[]);
```

25707 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 25708 themselves. In addition, the following variable:

```
25709 extern char **environ;
```

25710 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*  
 25711 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*  
 25712 array is not counted in *argc*.

25713 Conforming multi-threaded applications shall not use the *environ* variable to access or modify  
 25714 any environment variable while any other thread is concurrently modifying any environment  
 25715 variable. A call to any function dependent on any environment variable shall be considered a  
 25716 use of the *environ* variable to access that environment variable.

25717 The arguments specified by a program with one of the *exec* functions shall be passed on to the  
 25718 new process image in the corresponding *main()* arguments.

25719 The argument *path* points to a pathname that identifies the new process image file.

25720 The argument *file* is used to construct a pathname that identifies the new process image file. If  
 25721 the *file* argument contains a <slash> character, the *file* argument shall be used as the pathname  
 25722 for this file. Otherwise, the path prefix for this file is obtained by a search of the directories  
 25723 passed as the environment variable *PATH* (see XBD Chapter 8, on page 173). If this environment  
 25724 variable is not present, the results of the search are implementation-defined.

25725 There are two distinct ways in which the contents of the process image file may cause the  
 25726 execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the  
 25727 ERRORS section). In the cases where the other members of the *exec* family of functions would

25728 fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command  
 25729 interpreter and the environment of the executed command shall be as if the process invoked the  
 25730 *sh* utility using *execl()* as follows:

```
25731 execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

25732 where *<shell path>* is an unspecified pathname for the *sh* utility, *file* is the process image file, and  
 25733 for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv[0]*  
 25734 *argv[1]*, and so on.

25735 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These  
 25736 strings shall constitute the argument list available to the new process image. The list is  
 25737 terminated by a null pointer. The argument *arg0* should point to a filename that is associated  
 25738 with the process being started by one of the *exec* functions.

25739 The argument *argv* is an array of character pointers to null-terminated strings. The application  
 25740 shall ensure that the last member of this array is a null pointer. These strings shall constitute the  
 25741 argument list available to the new process image. The value in *argv[0]* should point to a filename  
 25742 that is associated with the process being started by one of the *exec* functions.

25743 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
 25744 shall constitute the environment for the new process image. The *envp* array is terminated by a  
 25745 null pointer.

25746 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the  
 25747 environment for the new process image shall be taken from the external variable *environ* in the  
 25748 calling process.

25749 The number of bytes available for the new process' combined argument and environment lists is  
 25750 {ARG\_MAX}. It is implementation-defined whether null terminators, pointers, and/or any  
 25751 alignment bytes are included in this total.

25752 File descriptors open in the calling process image shall remain open in the new process image,  
 25753 except for those whose close-on-exec flag FD\_CLOEXEC is set. For those file descriptors that  
 25754 remain open, all attributes of the open file description remain unchanged. For any file descriptor  
 25755 that is closed for this reason, file locks are removed as a result of the close as described in *close()*.  
 25756 Locks that are not removed by closing of file descriptors remain unchanged.

25757 If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec*  
 25758 family of functions, and the new process image file has the set-user-ID or set-group-ID file mode  
 25759 bits set, and the ST\_NOSUID bit is not set for the file system containing the new process image  
 25760 file, implementations may open an unspecified file for each of these file descriptors in the new  
 25761 process image.

25762 Directory streams open in the calling process image shall be closed in the new process image.

25763 The state of the floating-point environment in the initial thread of the new process image shall  
 25764 be set to the default.

25765 The state of conversion descriptors and message catalog descriptors in the new process image is  
 25766 undefined.

25767 For the new process image, the equivalent of:

```
25768 setlocale(LC_ALL, "C")
```

25769 shall be executed at start-up.

25770 Signals set to the default action (SIG\_DFL) in the calling process image shall be set to the default  
 25771 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG\_IGN) by

- 25772 the calling process image shall be set to be ignored by the new process image. Signals set to be  
 25773 caught by the calling process image shall be set to the default action in the new process image  
 25774 (see <signal.h>).
- 25775 If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether  
 25776 the SIGCHLD signal is set to be ignored or to the default action in the new process image.
- 25777 XSI After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and  
 25778 the SA\_ONSTACK flag shall be cleared for all signals.
- 25779 After a successful call to any of the *exec* functions, any functions previously registered by the  
 25780 *atexit()* or *pthread\_atfork()* functions are no longer registered.
- 25781 XSI If the ST\_NOSUID bit is set for the file system containing the new process image file, then the  
 25782 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged  
 25783 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is  
 25784 set, the effective user ID of the new process image shall be set to the user ID of the new process  
 25785 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the  
 25786 effective group ID of the new process image shall be set to the group ID of the new process  
 25787 image file. The real user ID, real group ID, and supplementary group IDs of the new process  
 25788 image shall remain the same as those of the calling process image. The effective user ID and  
 25789 effective group ID of the new process image shall be saved (as the saved set-user-ID and the  
 25790 saved set-group-ID) for use by *setuid()*.
- 25791 XSI Any shared memory segments attached to the calling process image shall not be attached to the  
 25792 new process image.
- 25793 Any named semaphores open in the calling process shall be closed as if by appropriate calls to  
 25794 *sem\_close()*.
- 25795 TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if  
 25796 *munmap()* was implicitly called to unmap them.
- 25797 ML Memory locks established by the calling process via calls to *mlockall()* or *mlock()* shall be  
 25798 removed. If locked pages in the address space of the calling process are also mapped into the  
 25799 address spaces of other processes and are locked by those processes, the locks established by the  
 25800 other processes shall be unaffected by the call by this process to the *exec* function. If the *exec*  
 25801 function fails, the effect on memory locks is unspecified.
- 25802 Memory mappings created in the process are unmapped before the address space is rebuilt for  
 25803 the new process image.
- 25804 SS When the calling process image does not use the SCHED\_FIFO, SCHED\_RR, or  
 25805 SCHED\_SPORADIC scheduling policies, the scheduling policy and parameters of the new  
 25806 process image and the initial thread in that new process image are implementation-defined.
- 25807 PS When the calling process image uses the SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC  
 25808 scheduling policies, the process policy and scheduling parameter settings shall not be changed  
 25809 by a call to an *exec* function. The initial thread in the new process image shall inherit the process  
 25810 scheduling policy and parameters. It shall have the default system contention scope, but shall  
 25811 inherit its allocation domain from the calling process image.
- 25812 Per-process timers created by the calling process shall be deleted before replacing the current  
 25813 process image with the new process image.
- 25814 MSG All open message queue descriptors in the calling process shall be closed, as described in  
 25815 *mq\_close()*.
- 25816 Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O

- 25817 operations that are not canceled shall complete as if the *exec* function had not yet occurred, but  
 25818 any associated signal notifications shall be suppressed. It is unspecified whether the *exec*  
 25819 function itself blocks awaiting such I/O completion. In no event, however, shall the new process  
 25820 image created by the *exec* function be affected by the presence of outstanding asynchronous I/O  
 25821 operations at the time the *exec* function is called. Whether any I/O is canceled, and which I/O  
 25822 may be canceled upon *exec*, is implementation-defined.
- 25823 CPT The new process image shall inherit the CPU-time clock of the calling process image. This  
 25824 inheritance means that the process CPU-time clock of the process being *exec*-ed shall not be  
 25825 reinitialized or altered as a result of the *exec* function other than to reflect the time spent by the  
 25826 process executing the *exec* function itself.
- 25827 TCT The initial value of the CPU-time clock of the initial thread of the new process image shall be set  
 25828 to zero.
- 25829 OB TRC If the calling process is being traced, the new process image shall continue to be traced into the  
 25830 same trace stream as the original process image, but the new process image shall not inherit the  
 25831 mapping of trace event names to trace event type identifiers that was defined by calls to the  
 25832 *posix\_trace\_eventid\_open()* or the *posix\_trace\_trid\_eventid\_open()* functions in the calling process  
 25833 image.
- 25834 If the calling process is a trace controller process, any trace streams that were created by the  
 25835 calling process shall be shut down as described in the *posix\_trace\_shutdown()* function.
- 25836 The thread ID of the initial thread in the new process image is unspecified.
- 25837 The size and location of the stack on which the initial thread in the new process image runs is  
 25838 unspecified.
- 25839 The initial thread in the new process image shall have its cancellation type set to  
 25840 `PTHREAD_CANCEL_DEFERRED` and its cancellation state set to  
 25841 `PTHREAD_CANCEL_ENABLED`.
- 25842 The initial thread in the new process image shall have all thread-specific data values set to  
 25843 `NULL` and all thread-specific data keys shall be removed by the call to *exec* without running  
 25844 destructors.
- 25845 The initial thread in the new process image shall be joinable, as if created with the *detachstate*  
 25846 attribute set to `PTHREAD_CREATE_JOINABLE`.
- 25847 The new process shall inherit at least the following attributes from the calling process image:
- 25848 XSI • Nice value (see *nice()*)
  - 25849 XSI • *semadj* values (see *semop()*)
  - 25850 • Process ID
  - 25851 • Parent process ID
  - 25852 • Process group ID
  - 25853 • Session membership
  - 25854 • Real user ID
  - 25855 • Real group ID
  - 25856 • Supplementary group IDs

- 25857 • Time left until an alarm clock signal (see *alarm()*)
- 25858 • Current working directory
- 25859 • Root directory
- 25860 • File mode creation mask (see *umask()*)
- 25861 XSI • File size limit (see *getrlimit()* and *setrlimit()*)
- 25862 • Process signal mask (see *pthread\_sigmask()*)
- 25863 • Pending signal (see *sigpending()*)
- 25864 • *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime* (see *times()*)
- 25865 XSI • Resource limits
- 25866 • Controlling terminal
- 25867 XSI • Interval timers
- 25868 The initial thread of the new process shall inherit at least the following attributes from the  
25869 calling thread:
- 25870 • Signal mask (see *sigprocmask()* and *pthread\_sigmask()*)
- 25871 • Pending signals (see *sigpending()*)
- 25872 All other process attributes defined in this volume of POSIX.1-2008 shall be inherited in the new  
25873 process image from the old process image. All other thread attributes defined in this volume of  
25874 POSIX.1-2008 shall be inherited in the initial thread in the new process image from the calling  
25875 thread in the old process image. The inheritance of process or thread attributes not defined by  
25876 this volume of POSIX.1-2008 is implementation-defined.
- 25877 A call to any *exec* function from a process with more than one thread shall result in all threads  
25878 being terminated and the new executable image being loaded and executed. No destructor  
25879 functions or cleanup handlers shall be called.
- 25880 Upon successful completion, the *exec* functions shall mark for update the last data access  
25881 timestamp of the file. If an *exec* function failed but was able to locate the process image file,  
25882 whether the last data access timestamp is marked for update is unspecified. Should the *exec*  
25883 function succeed, the process image file shall be considered to have been opened with *open()*.  
25884 The corresponding *close()* shall be considered to occur at a time after this open, but before  
25885 process termination or successful completion of a subsequent call to one of the *exec* functions,  
25886 *posix\_spawn()*, or *posix\_spawnp()*. The *argv[]* and *envp[]* arrays of pointers and the strings to  
25887 which those arrays point shall not be modified by a call to one of the *exec* functions, except as a  
25888 consequence of replacing the process image.
- 25889 XSI The saved resource limits in the new process image are set to be a copy of the process'  
25890 corresponding hard and soft limits.
- 25891 **RETURN VALUE**
- 25892 If one of the *exec* functions returns to the calling process image, an error has occurred; the return  
25893 value shall be  $-1$ , and *errno* shall be set to indicate the error.
- 25894 **ERRORS**
- 25895 The *exec* functions shall fail if:
- 25896 [E2BIG] The number of bytes used by the new process image's argument list and  
25897 environment list is greater than the system-imposed limit of {ARG\_MAX}  
25898 bytes.

|       |                |                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 25899 | [EACCES]       | Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.                                                                                               |
| 25900 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25901 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25902 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25903 | [EINVAL]       | The new process image file has appropriate privileges and has a recognized executable binary format, but the system does not support execution of a file with this format.                                                                                                                                                                                                           |
| 25904 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25905 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25906 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i> or <i>file</i> argument.                                                                                                                                                                                                                                                                            |
| 25907 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25908 | [ENAMETOOLONG] |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25909 |                | The length of a component of a pathname is longer than {NAME_MAX}.                                                                                                                                                                                                                                                                                                                   |
| 25910 | [ENOENT]       | A component of <i>path</i> or <i>file</i> does not name an existing file or <i>path</i> or <i>file</i> is an empty string.                                                                                                                                                                                                                                                           |
| 25911 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25912 | [ENOTDIR]      | A component of the new process image file's path prefix is not a directory, or the new process image file's pathname contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory. |
| 25913 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25914 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25915 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25916 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25917 |                | The <i>exec</i> functions, except for <i>execlp()</i> and <i>execvp()</i> , shall fail if:                                                                                                                                                                                                                                                                                           |
| 25918 | [ENOEXEC]      | The new process image file has the appropriate access permission but has an unrecognized format.                                                                                                                                                                                                                                                                                     |
| 25919 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25920 |                | The <i>fexecve()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                         |
| 25921 | [EBADF]        | The <i>fd</i> argument is not a valid file descriptor open for executing.                                                                                                                                                                                                                                                                                                            |
| 25922 |                | The <i>exec</i> functions may fail if:                                                                                                                                                                                                                                                                                                                                               |
| 25923 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> or <i>file</i> argument.                                                                                                                                                                                                                                                                |
| 25924 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25925 | [ENAMETOOLONG] |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25926 |                | The length of the <i>path</i> argument or the length of the pathname constructed from the <i>file</i> argument exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.                                                                                                                                  |
| 25927 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25928 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25929 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25930 | [ENOMEM]       | The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.                                                                                                                                                                                                                                                          |
| 25931 |                |                                                                                                                                                                                                                                                                                                                                                                                      |
| 25932 | [ETXTBSY]      | The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.                                                                                                                                                                                                                                                                |
| 25933 |                |                                                                                                                                                                                                                                                                                                                                                                                      |

## 25934 EXAMPLES

25935 **Using execl()**

25936 The following example executes the *ls* command, specifying the pathname of the executable  
25937 (*/bin/ls*) and using arguments supplied directly to the command to produce single-column  
25938 output.

```
25939 #include <unistd.h>
25940 int ret;
25941 ...
25942 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

25943 **Using execl\_e()**

25944 The following example is similar to [Using execl\(\)](#). In addition, it specifies the environment for  
25945 the new process image using the *env* argument.

```
25946 #include <unistd.h>
25947 int ret;
25948 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
25949 ...
25950 ret = execl_e ("/bin/ls", "ls", "-l", (char *)0, env);
```

25951 **Using execl\_p()**

25952 The following example searches for the location of the *ls* command among the directories  
25953 specified by the *PATH* environment variable.

```
25954 #include <unistd.h>
25955 int ret;
25956 ...
25957 ret = execl_p ("ls", "ls", "-l", (char *)0);
```

25958 **Using execv()**

25959 The following example passes arguments to the *ls* command in the *cmd* array.

```
25960 #include <unistd.h>
25961 int ret;
25962 char *cmd[] = { "ls", "-l", (char *)0 };
25963 ...
25964 ret = execv ("/bin/ls", cmd);
```

25965 **Using `execve()`**

25966 The following example passes arguments to the `ls` command in the `cmd` array, and specifies the  
25967 environment for the new process image using the `env` argument.

```
25968 #include <unistd.h>
25969
25969 int ret;
25970 char *cmd[] = { "ls", "-l", (char *)0 };
25971 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
25972 ...
25973 ret = execve ("/bin/ls", cmd, env);
```

25974 **Using `execvp()`**

25975 The following example searches for the location of the `ls` command among the directories  
25976 specified by the `PATH` environment variable, and passes arguments to the `ls` command in the  
25977 `cmd` array.

```
25978 #include <unistd.h>
25979
25979 int ret;
25980 char *cmd[] = { "ls", "-l", (char *)0 };
25981 ...
25982 ret = execvp ("ls", cmd);
```

25983 **APPLICATION USAGE**

25984 As the state of conversion descriptors and message catalog descriptors in the new process image  
25985 is undefined, conforming applications should not rely on their use and should close them prior  
25986 to calling one of the `exec` functions.

25987 Applications that require other than the default POSIX locale should call `setlocale()` with the  
25988 appropriate parameters to establish the locale of the new process.

25989 The `environ` array should not be accessed directly by the application.

25990 The new process might be invoked in a non-conforming environment if the `envp` array does not  
25991 contain implementation-defined variables required by the implementation to provide a  
25992 conforming environment. See the `_CS_V7_ENV` entry in `<unistd.h>` and `confstr()` for details.

25993 Applications should not depend on file descriptors 0, 1, and 2 being closed after an `exec`. A  
25994 future version may allow these file descriptors to be automatically opened for any process.

25995 If an application wants to perform a checksum test of the file being executed before executing it,  
25996 the file will need to be opened with read permission to perform the checksum test.

25997 Since execute permission is checked by `fexecve()`, the file description `fd` need not have been  
25998 opened with the `O_EXEC` flag. However, if the file to be executed denies read and write  
25999 permission for the process preparing to do the `exec`, the only way to provide the `fd` to `fexecve()`  
26000 will be to use the `O_EXEC` flag when opening `fd`. In this case, the application will not be able to  
26001 perform a checksum test since it will not be able to read the contents of the file.

26002 Note that when a file descriptor is opened with `O_RDONLY`, `O_RDWR`, or `O_WRONLY` mode,  
26003 the file descriptor can be used to read, read and write, or write the file, respectively, even if the  
26004 mode of the file changes after the file was opened. Using the `O_EXEC` open mode is different;  
26005 `fexecve()` will ignore the mode that was used when the file descriptor was opened and the `exec`  
26006 will fail if the mode of the file associated with `fd` does not grant execute permission to the calling  
26007 process at the time `fexecve()` is called.

## 26008 RATIONALE

26009 Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was  
 26010 driven by the same requirement in drafts of the ISO C standard. In fact, historical  
 26011 implementations have passed a value of zero when no arguments are supplied to the caller of  
 26012 the *exec* functions. This requirement was removed from the ISO C standard and subsequently  
 26013 removed from this volume of POSIX.1-2008 as well. The wording, in particular the use of the  
 26014 word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to  
 26015 the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an  
 26016 application. In fact, this is good practice, since many existing applications reference *argv[0]*  
 26017 without first checking the value of *argc*.

26018 The requirement on a Strictly Conforming POSIX Application also states that the value passed as  
 26019 the first argument be a filename associated with the process being started. Although some  
 26020 existing applications pass a pathname rather than a filename in some circumstances, a filename  
 26021 is more generally useful, since the common usage of *argv[0]* is in printing diagnostics. In some  
 26022 cases the filename passed is not the actual filename of the file, for example, many  
 26023 implementations of the *login* utility use a convention of prefixing a <hyphen> (‘-’) to the actual  
 26024 filename, which indicates to the command interpreter being invoked that it is a “login shell”.

26025 Historically, there have been two ways that implementations can *exec* shell scripts.

26026 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*  
 26027 functions return an [ENOEXEC] error for any file not recognizable as executable, including a  
 26028 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file  
 26029 to be a shell script and invoke a known command interpreter to interpret such files. This is now  
 26030 required by POSIX.1-2008. These implementations of *execvp()* and *execlp()* only give the  
 26031 [ENOEXEC] error in the rare case of a problem with the command interpreter’s executable file.  
 26032 Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or  
 26033 *execvp()*, although implementations can still give it.

26034 Another way that some historical implementations handle shell scripts is by recognizing the first  
 26035 two bytes of the file as the character string “#!” and using the remainder of the first line of the  
 26036 file as the name of the command interpreter to execute.

26037 One potential source of confusion noted by the standard developers is over how the contents of  
 26038 a process image file affect the behavior of the *exec* family of functions. The following is a  
 26039 description of the actions taken:

- 26040 1. If the process image file is a valid executable (in a format that is executable and valid and  
 26041 having appropriate privileges) for this system, then the system executes the file.
- 26042 2. If the process image file has appropriate privileges and is in a format that is executable  
 26043 but not valid for this system (such as a recognized binary for another architecture), then  
 26044 this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
- 26045 3. If the process image file has appropriate privileges but is not otherwise recognized:
  - 26046 a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter  
 26047 assuming that the process image file is a shell script.
  - 26048 b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to  
 26049 [ENOEXEC].

26050 Applications that do not require to access their arguments may use the form:

26051 `main(void)`

26052 as specified in the ISO C standard. However, the implementation will always provide the two

26053 arguments *argc* and *argv*, even if they are not used.

26054 Some implementations provide a third argument to *main()* called *envp*. This is defined as a  
26055 pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,  
26056 so implementations must support applications written this way. Since this volume of  
26057 POSIX.1-2008 defines the global variable *environ*, which is also provided by historical  
26058 implementations and can be used anywhere that *envp* could be used, there is no functional need  
26059 for the *envp* argument. Applications should use the *getenv()* function rather than accessing the  
26060 environment directly via either *envp* or *environ*. Implementations are required to support the  
26061 two-argument calling sequence, but this does not prohibit an implementation from supporting  
26062 *envp* as an optional third argument.

26063 This volume of POSIX.1-2008 specifies that signals set to SIG\_IGN remain set to SIG\_IGN, and  
26064 that the new process image inherits the signal mask of the thread that called *exec* in the old  
26065 process image. This is consistent with historical implementations, and it permits some useful  
26066 functionality, such as the *nohup* command. However, it should be noted that many existing  
26067 applications wrongly assume that they start with certain signals set to the default action and/or  
26068 unblocked. In particular, applications written with a simpler signal model that does not include  
26069 blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked  
26070 with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without  
26071 explicit reason to do so, and especially not to block signals across *execs* of arbitrary (not closely  
26072 co-operating) programs.

26073 The *exec* functions always save the value of the effective user ID and effective group ID of the  
26074 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of  
26075 the process image file is set.

26076 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future  
26077 writers of language bindings that these objects are completely constant. Due to a limitation of  
26078 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of  
26079 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the  
26080 natural choice, given that these functions do not modify either the array of pointers or the  
26081 characters to which the function points, but this would disallow existing correct code. Instead,  
26082 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*  
26083 derived from the ISO C standard summarizes the compatibility:

| 26084 | <i>dst:</i>         | char *[] | const char *[] | char *const[] | const char *const[] |
|-------|---------------------|----------|----------------|---------------|---------------------|
| 26085 | <i>src:</i>         |          |                |               |                     |
| 26086 | char *[]            | VALID    | —              | VALID         | —                   |
| 26087 | const char *[]      | —        | VALID          | —             | VALID               |
| 26088 | char *const[]       | —        | —              | VALID         | —                   |
| 26089 | const char *const[] | —        | —              | —             | VALID               |

26090 Since all existing code has a source type matching the first row, the column that gives the most  
26091 valid combinations is the third column. The only other possibility is the fourth column, but  
26092 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth  
26093 column cannot be used, because the declaration a non-expert would naturally use would be that  
26094 in the second row.

26095 The ISO C standard and this volume of POSIX.1-2008 do not conflict on the use of *environ*, but  
26096 some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in  
26097 the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can  
26098 contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no  
26099 problem ensues. The situation is similar for *environ*: the definition in this volume of  
26100 POSIX.1-2008 is to be used if there is no user-provided *environ* to take precedence. At least three

|       |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26101 |                         | implementations are known to exist that solve this problem.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 26102 | [E2BIG]                 | The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 26103 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26104 | [EFAULT]                | Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 26105 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26106 | [EINVAL]                | This error condition was added to POSIX.1-2008 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, <i>execvp()</i> , and <i>execlp()</i> that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the <i>path</i> argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons: |
| 26107 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26108 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26109 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26110 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26111 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26112 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26113 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26114 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26115 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26116 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26117 |                         | 1. The new utilization of [EINVAL] will provide some measure of utility to the user community.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 26118 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26119 |                         | 2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 26120 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26121 | [ENAMETOOLONG]          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26122 |                         | Since the file pathname may be constructed by taking elements in the <i>PATH</i> variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 26123 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26124 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26125 | [ETXTBSY]               | System V returns this error when the executable file is currently open for writing by some process. This volume of POSIX.1-2008 neither requires nor prohibits this behavior.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 26126 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26127 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26128 |                         | Other systems (such as System V) may return [EINTR] from <i>exec</i> . This is not addressed by this volume of POSIX.1-2008, but implementations may have a window between the call to <i>exec</i> and the time that a signal could cause one of the <i>exec</i> calls to return with [EINTR].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 26129 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26130 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26131 |                         | An explicit statement regarding the floating-point environment (as defined in the <code>&lt;fenv.h&gt;</code> header) was added to make it clear that the floating-point environment is set to its default when a call to one of the <i>exec</i> functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:                                                                                                                                                                                                                                                                                                                                                                                                           |
| 26132 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26133 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26134 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26135 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26136 | <i>posix_spawn()</i>    | Set to default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26137 | <i>fork()</i>           | Inherit.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26138 | <i>pthread_create()</i> | Inherit.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26139 |                         | The purpose of the <i>fexecve()</i> function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, an function like <i>openat()</i> can be used to open a file which has been found by reading the content of a directory using <i>readdir()</i> .                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 26140 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26141 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26142 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 26143 |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

26144 **FUTURE DIRECTIONS**

26145 None.

26146 **SEE ALSO**

26147 *alarm()*, *atexit()*, *chmod()*, *close()*, *confstr()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*,  
 26148 *getrlimit()*, *mknod()*, *mmap()*, *nice()*, *open()*, *posix\_spawn()*, *posix\_trace\_create()*,  
 26149 *posix\_trace\_event()*, *posix\_trace\_eventid\_equal()*, *pthread\_atfork()*, *pthread\_sigmask()*, *putenv()*,  
 26150 *readdir()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *system()*, *times()*,  
 26151 *ulimit()*, *umask()*

26152 XBD Chapter 8 (on page 173), [<unistd.h>](#)26153 **CHANGE HISTORY**

26154 First released in Issue 1. Derived from Issue 1 of the SVID.

26155 **Issue 5**26156 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
26157 Threads Extension.

26158 Large File Summit extensions are added.

26159 **Issue 6**26160 The following new requirements on POSIX implementations derive from alignment with the  
26161 Single UNIX Specification:

- 26162 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid  
26163 executable.
- 26164 • In this version, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective  
26165 group ID of the new process image shall be saved (as the saved set-user-ID and the saved  
26166 set-group-ID) for use by the `setuid(0)` function.
- 26167 • The [ELOOP] mandatory error condition is added.
- 26168 • A second [ENAMETOOLONG] is added as an optional error condition.
- 26169 • The [ETXTBSY] optional error condition is added.

26170 The following changes were made to align with the IEEE P1003.1a draft standard:

- 26171 • The [EINVAL] mandatory error condition is added.
- 26172 • The [ELOOP] optional error condition is added.

26173 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

26174 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
26175 for typed memory.

26176 The normative text is updated to avoid use of the term “must” for application requirements.

26177 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

26178 IEEE PASC Interpretation 1003.1 #132 is applied.

26179 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new  
26180 process image is set to the default.26181 The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents  
26182 of a process image file affect the behavior of the *exec* functions.26183 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to  
26184 the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change

- 26185 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,  
26186 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of  
26187 functions.
- 26188 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the  
26189 DESCRIPTION, addressing which attributes are inherited by threads, and behavioral  
26190 requirements for threads attributes.
- 26191 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the  
26192 RATIONALE from “the process signal mask be unchanged across an *exec*” to “the new process  
26193 image inherits the signal mask of the thread that called *exec* in the old process image”.
- 26194 **Issue 7**
- 26195 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of `_CS_V7_ENV`  
26196 to the APPLICATION USAGE.
- 26197 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 26198 The *fexecve()* function is added from The Open Group Technical Standard, 2006, Extended API  
26199 Set Part 2.
- 26200 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads,  
26201 and Timers options is moved to the Base.
- 26202 Changes are made related to support for finegrained timestamps.

26203 **NAME**

26204 exit — terminate a process

26205 **SYNOPSIS**

26206 #include &lt;stdlib.h&gt;

26207 void exit(int status);

26208 **DESCRIPTION**

26209 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26210 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26211 volume of POSIX.1-2008 defers to the ISO C standard.

26212 CX The value of *status* may be 0, EXIT\_SUCCESS, EXIT\_FAILURE, or any other value, though only  
 26213 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

26214 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their  
 26215 registration, except that a function is called after any previously registered functions that had  
 26216 already been called at the time it was registered. Each function is called as many times as it was  
 26217 registered. If, during the call to any such function, a call to the *longjmp()* function is made that  
 26218 would terminate the call to the registered function, the behavior is undefined.

26219 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall  
 26220 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more  
 26221 than once, the behavior is undefined.

26222 The *exit()* function shall then flush all open streams with unwritten buffered data and close all  
 26223 CX open streams. Finally, the process shall be terminated with the same consequences as described  
 26224 in [Consequences of Process Termination](#) (on page 545).

26225 **RETURN VALUE**26226 The *exit()* function does not return.26227 **ERRORS**

26228 No errors are defined.

26229 **EXAMPLES**

26230 None.

26231 **APPLICATION USAGE**

26232 None.

26233 **RATIONALE**26234 See *\_Exit()*.26235 **FUTURE DIRECTIONS**

26236 None.

26237 **SEE ALSO**26238 *\_Exit()*, *atexit()*, *exec*, *longjmp()*, *tmpfile()*

26239 XBD &lt;stdlib.h&gt;

26240 **CHANGE HISTORY**26241 **Issue 7**26242 Austin Group Interpretation 1003.1-2001 #031 is applied, separating the *\_Exit()* and *\_exit()*  
 26243 functions from the *exit()* function.

26244 Austin Group Interpretation 1003.1-2001 #085 is applied.

**exp()**26245 **NAME**26246 `exp, expf, expl` — exponential function26247 **SYNOPSIS**26248 `#include <math.h>`26249 `double exp(double x);`26250 `float expf(float x);`26251 `long double expl(long double x);`26252 **DESCRIPTION**26253 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
26254 conflict between the requirements described here and the ISO C standard is unintentional. This  
26255 volume of POSIX.1-2008 defers to the ISO C standard.26256 These functions shall compute the base-*e* exponential of *x*.26257 An application wishing to check for error situations should set *errno* to zero and call  
26258 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
26259 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
26260 zero, an error has occurred.26261 **RETURN VALUE**26262 Upon successful completion, these functions shall return the exponential value of *x*.26263 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*  
26264 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.26265 If the correct value would cause underflow, and is not representable, a range error may occur,  
26266 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.26267 **MX** If *x* is NaN, a NaN shall be returned.26268 If *x* is ±0, 1 shall be returned.26269 If *x* is -Inf, +0 shall be returned.26270 If *x* is +Inf, *x* shall be returned.26271 If the correct value would cause underflow, and is representable, a range error may occur and  
26272 the correct value shall be returned.26273 **ERRORS**

26274 These functions shall fail if:

26275 **Range Error** The result overflows.26276 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
26277 then *errno* shall be set to [ERANGE]. If the integer expression  
26278 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
26279 floating-point exception shall be raised.

26280 These functions may fail if:

26281 **Range Error** The result underflows.26282 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
26283 then *errno* shall be set to [ERANGE]. If the integer expression  
26284 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
26285 floating-point exception shall be raised.

26286 **EXAMPLES**26287 **Computing the Density of the Standard Normal Distribution**

26288 This function shows an implementation for the density of the standard normal distribution  
 26289 using *exp()*. This example uses the constant `M_PI` which is part of the XSI option.

```
26290 #include <math.h>
26291
26292 double
26293 normal_density (double x)
26294 {
26295     return exp(-x*x/2) / sqrt (2*M_PI);
26296 }
```

26296 **APPLICATION USAGE**

26297 Note that for IEEE Std 754-1985 **double**,  $709.8 < x$  implies *exp(x)* has overflowed. The value  
 26298  $x < -708.4$  implies *exp(x)* has underflowed.

26299 On error, the expressions (*math\_errhandling* & `MATH_ERRNO`) and (*math\_errhandling* &  
 26300 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

26301 **RATIONALE**

26302 None.

26303 **FUTURE DIRECTIONS**

26304 None.

26305 **SEE ALSO**

26306 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#)

26307 XBD Section 4.19 (on page 116), [<math.h>](#)

26308 **CHANGE HISTORY**

26309 First released in Issue 1. Derived from Issue 1 of the SVID.

26310 **Issue 5**

26311 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 26312 text was previously published in the APPLICATION USAGE section.

26313 **Issue 6**

26314 The *expf()* and *expl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

26315 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 26316 revised to align with the ISO/IEC 9899:1999 standard.

26317 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 26318 marked.

26319 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the  
 26320 EXAMPLES section.

**exp2()**26321 **NAME**26322 `exp2, exp2f, exp2l` — exponential base 2 functions26323 **SYNOPSIS**

```
26324 #include <math.h>
26325
26325 double exp2(double x);
26326 float exp2f(float x);
26327 long double exp2l(long double x);
```

26328 **DESCRIPTION**

26329 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 26330 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26331 volume of POSIX.1-2008 defers to the ISO C standard.

26332 These functions shall compute the base-2 exponential of  $x$ .

26333 An application wishing to check for error situations should set *errno* to zero and call  
 26334 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26335 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26336 zero, an error has occurred.

26337 **RETURN VALUE**

26338 Upon successful completion, these functions shall return  $2^x$ .

26339 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and  
 26340 *exp2l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 26341 respectively.

26342 If the correct value would cause underflow, and is not representable, a range error may occur,  
 26343 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

26344 **MX** If  $x$  is NaN, a NaN shall be returned.

26345 If  $x$  is  $\pm 0$ , 1 shall be returned.

26346 If  $x$  is  $-\text{Inf}$ , +0 shall be returned.

26347 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

26348 If the correct value would cause underflow, and is representable, a range error may occur and  
 26349 the correct value shall be returned.

26350 **ERRORS**

26351 These functions shall fail if:

26352 **Range Error** The result overflows.

26353 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26354 then *errno* shall be set to [ERANGE]. If the integer expression  
 26355 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 26356 floating-point exception shall be raised.

26357 These functions may fail if:

26358 **Range Error** The result underflows.

26359 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26360 then *errno* shall be set to [ERANGE]. If the integer expression  
 26361 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 26362 floating-point exception shall be raised.

26363 **EXAMPLES**

26364 None.

26365 **APPLICATION USAGE**26366 For IEEE Std 754-1985 **double**,  $1024 \leq x$  implies  $\text{exp2}(x)$  has overflowed. The value  $x < -1022$   
26367 implies  $\text{exp}(x)$  has underflowed.26368 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
26369 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.26370 **RATIONALE**

26371 None.

26372 **FUTURE DIRECTIONS**

26373 None.

26374 **SEE ALSO**26375 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*26376 XBD Section 4.19 (on page 116), **<math.h>**26377 **CHANGE HISTORY**

26378 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**expm1()**26379 **NAME**26380 `expm1, expm1f, expm1l` — compute exponential functions26381 **SYNOPSIS**

```
26382 #include <math.h>
26383 double expm1(double x);
26384 float expm1f(float x);
26385 long double expm1l(long double x);
```

26386 **DESCRIPTION**

26387 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 26388 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26389 volume of POSIX.1-2008 defers to the ISO C standard.

26390 These functions shall compute  $e^x - 1.0$ .

26391 An application wishing to check for error situations should set *errno* to zero and call  
 26392 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26393 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26394 zero, an error has occurred.

26395 **RETURN VALUE**26396 Upon successful completion, these functions return  $e^x - 1.0$ .

26397 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and  
 26398 *expm1l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 26399 respectively.

26400 **MX** If *x* is NaN, a NaN shall be returned.26401 If *x* is  $\pm 0$ ,  $\pm 0$  shall be returned.26402 If *x* is  $-\text{Inf}$ ,  $-1$  shall be returned.26403 If *x* is  $+\text{Inf}$ , *x* shall be returned.26404 If *x* is subnormal, a range error may occur and *x* should be returned.26405 **ERRORS**

26406 These functions shall fail if:

26407 **Range Error** The result overflows.

26408 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26409 then *errno* shall be set to [ERANGE]. If the integer expression  
 26410 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 26411 floating-point exception shall be raised.

26412 These functions may fail if:

26413 **MX** **Range Error** The value of *x* is subnormal.

26414 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26415 then *errno* shall be set to [ERANGE]. If the integer expression  
 26416 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 26417 floating-point exception shall be raised.

26418 **EXAMPLES**

26419 None.

26420 **APPLICATION USAGE**26421 The value of  $\text{expm1}(x)$  may be more accurate than  $\text{exp}(x)-1.0$  for small values of  $x$ .26422 The  $\text{expm1}()$  and  $\text{log1p}()$  functions are useful for financial calculations of  $((1+x)^n-1)/x$ , namely:26423  $\text{expm1}(n * \text{log1p}(x)) / x$ 26424 when  $x$  is very small (for example, when calculating small daily interest rates). These functions  
26425 also simplify writing accurate inverse hyperbolic functions.26426 For IEEE Std 754-1985 **double**,  $709.8 < x$  implies  $\text{expm1}(x)$  has overflowed.26427 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
26428 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.26429 **RATIONALE**

26430 None.

26431 **FUTURE DIRECTIONS**

26432 None.

26433 **SEE ALSO**26434 *exp()*, *feclearexcept()*, *fetestexcept()*, *ilogb()*, *log1p()*26435 XBD Section 4.19 (on page 116), **<math.h>**26436 **CHANGE HISTORY**

26437 First released in Issue 4, Version 2.

26438 **Issue 5**

26439 Moved from X/OPEN UNIX extension to BASE.

26440 **Issue 6**26441 The  $\text{expm1f}()$  and  $\text{expm1l}()$  functions are added for alignment with the ISO/IEC 9899:1999  
26442 standard.26443 The  $\text{expm1}()$  function is no longer marked as an extension.26444 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
26445 revised to align with the ISO/IEC 9899:1999 standard.26446 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
26447 marked.

**fabs()**26448 **NAME**26449 `fabs, fabsf, fabsl` — absolute value function26450 **SYNOPSIS**

```
26451 #include <math.h>
26452 double fabs(double x);
26453 float fabsf(float x);
26454 long double fabsl(long double x);
```

26455 **DESCRIPTION**

26456 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 26457 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26458 volume of POSIX.1-2008 defers to the ISO C standard.

26459 These functions shall compute the absolute value of their argument  $x$ ,  $|x|$ .

26460 **RETURN VALUE**

26461 Upon successful completion, these functions shall return the absolute value of  $x$ .

26462 **MX** If  $x$  is NaN, a NaN shall be returned.

26463 If  $x$  is  $\pm 0$ ,  $+0$  shall be returned.

26464 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

26465 **ERRORS**

26466 No errors are defined.

26467 **EXAMPLES**26468 **Computing the 1-Norm of a Floating-Point Vector**

26469 This example shows the use of `fabs()` to compute the 1-norm of a vector defined as follows:

```
26470 norm1(v) = |v[0]| + |v[1]| + ... + |v[n-1]|
```

26471 where  $|x|$  denotes the absolute value of  $x$ ,  $n$  denotes the vector's dimension  $v[i]$  denotes the  $i$ -th  
 26472 component of  $v$  ( $0 \leq i < n$ ).

```
26473 #include <math.h>
26474 double
26475 norm1(const double v[], const int n)
26476 {
26477     int i;
26478     double n1_v; /* 1-norm of v */
26479     n1_v = 0;
26480     for (i=0; i<n; i++) {
26481         n1_v += fabs (v[i]);
26482     }
26483     return n1_v;
26484 }
```

26485 **APPLICATION USAGE**

26486 None.

26487 **RATIONALE**

26488 None.

26489 **FUTURE DIRECTIONS**

26490 None.

26491 **SEE ALSO**26492 [isnan\(\)](#)26493 XBD [<math.h>](#)26494 **CHANGE HISTORY**

26495 First released in Issue 1. Derived from Issue 1 of the SVID.

26496 **Issue 5**26497 The DESCRIPTION is updated to indicate how an application should check for an error. This  
26498 text was previously published in the APPLICATION USAGE section.26499 **Issue 6**26500 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.26501 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
26502 revised to align with the ISO/IEC 9899:1999 standard.26503 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
26504 marked.26505 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the  
26506 EXAMPLES section.

**faccessat()***System Interfaces*26507 **NAME**26508            `faccessat` — determine accessibility of a file relative to directory file descriptor26509 **SYNOPSIS**26510            `#include <unistd.h>`26511            `int faccessat(int fd, const char *path, int amode, int flag);`26512 **DESCRIPTION**26513            Refer to `access()`.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

26514 **NAME**

26515 **fattach** — attach a STREAMS-based file descriptor to a file in the file system name space  
 26516 (**STREAMS**)

26517 **SYNOPSIS**

```
26518 OB XSR #include <stropts.h>
26519 int fattach(int fildev, const char *path);
```

26520 **DESCRIPTION**

26521 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively  
 26522 associating a pathname with *fildev*. The application shall ensure that the *fildev* argument is a  
 26523 valid open file descriptor associated with a STREAMS file. The *path* argument points to a  
 26524 pathname of an existing file. The application shall have appropriate privileges or be the owner  
 26525 of the file named by *path* and have write permission. A successful call to *fattach()* shall cause all  
 26526 pathnames that name the file named by *path* to name the STREAMS file associated with *fildev*,  
 26527 until the STREAMS file is detached from the file. A STREAMS file can be attached to more than  
 26528 one file and can have several pathnames associated with it.

26529 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user  
 26530 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,  
 26531 and the size and device identifier are set to those of the STREAMS file associated with *fildev*. If  
 26532 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),  
 26533 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildev*  
 26534 refers shall be affected.

26535 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to  
 26536 refer to the underlying file.

26537 **RETURN VALUE**

26538 Upon successful completion, *fattach()* shall return 0. Otherwise,  $-1$  shall be returned and *errno*  
 26539 set to indicate the error.

26540 **ERRORS**

26541 The *fattach()* function shall fail if:

26542 [EACCES] Search permission is denied for a component of the path prefix, or the process  
 26543 is the owner of *path* but does not have write permissions on the file named by  
 26544 *path*.

26545 [EBADF] The *fildev* argument is not a valid open file descriptor.

26546 [EBUSY] The file named by *path* is currently a mount point or has a STREAMS file  
 26547 attached to it.

26548 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 26549 argument.

26550 [ENAMETOOLONG]

26551 The length of a component of a pathname is longer than {NAME\_MAX}.

26552 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

26553 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
 26554 contains at least one non- $\langle$ slash $\rangle$  character and ends with one or more trailing  
 26555  $\langle$ slash $\rangle$  characters and the last pathname component names an existing file  
 26556 that is neither a directory nor a symbolic link to a directory.

**fattach()**

- 26557 [EPERM] The effective user ID of the process is not the owner of the file named by *path*  
 26558 and the process does not have appropriate privileges.
- 26559 The *fattach()* function may fail if:
- 26560 [EINVAL] The *files* argument does not refer to a STREAMS file.
- 26561 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 26562 resolution of the *path* argument.
- 26563 [ENAMETOOLONG]  
 26564 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 26565 symbolic link produced an intermediate result with a length that exceeds  
 26566 {PATH\_MAX}.
- 26567 [EXDEV] A link to a file on another file system was attempted.

26568 **EXAMPLES**26569 **Attaching a File Descriptor to a File**

26570 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this  
 26571 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-**  
 26572 **STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file  
 26573 handle referring to the STREAMS file associated with *fd*.

```
26574 #include <stropts.h>
26575 ...
26576     int fd;
26577     char *filename = "/tmp/named-STREAM";
26578     int ret;
26579
26579     ret = fattach(fd, filename);
```

26580 **APPLICATION USAGE**

26581 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is  
 26582 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the  
 26583 replaced file need not be a directory and the replacing file is a STREAMS file.

26584 **RATIONALE**

26585 The file attributes of a file which has been the subject of an *fattach()* call are specifically set  
 26586 because of an artifact of the original implementation. The internal mechanism was the same as  
 26587 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when  
 26588 applied to a regular file are a little surprising, especially as regards the link count which rigidly  
 26589 remains one, even if there were several links originally and despite the fact that all original links  
 26590 refer to the STREAM as long as the *fattach()* remains in effect.

26591 **FUTURE DIRECTIONS**

26592 The *fattach()* function may be removed in a future version.

26593 **SEE ALSO**

26594 *fdetach()*, *isastream()*

26595 XBD <stropts.h>

26596 **CHANGE HISTORY**

26597 First released in Issue 4, Version 2.

26598 **Issue 5**

26599 Moved from X/OPEN UNIX extension to BASE.

26600 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

26601 **Issue 6**

26602 This function is marked as part of the XSI STREAMS Option Group.

26603 The normative text is updated to avoid use of the term “must” for application requirements.

26604 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
26605 [ELOOP] error condition is added.

26606 **Issue 7**

26607 Austin Group Interpretation 1003.1-2001 #143 is applied.

26608 The *fattach()* function is marked obsolescent.

26609 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
26610 pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fchdir()**26611 **NAME**26612 `fchdir` — change working directory26613 **SYNOPSIS**26614 `#include <unistd.h>`26615 `int fchdir(int fildev);`26616 **DESCRIPTION**26617 The `fchdir()` function shall be equivalent to `chdir()` except that the directory that is to be the new  
26618 current working directory is specified by the file descriptor `fildev`.26619 A conforming application can obtain a file descriptor for a file of type directory using `open()`,  
26620 provided that the file status flags and access modes do not contain `O_WRONLY` or `O_RDWR`.26621 **RETURN VALUE**26622 Upon successful completion, `fchdir()` shall return 0. Otherwise, it shall return -1 and set `errno` to  
26623 indicate the error. On failure the current working directory shall remain unchanged.26624 **ERRORS**26625 The `fchdir()` function shall fail if:26626 [EACCES] Search permission is denied for the directory referenced by `fildev`.26627 [EBADF] The `fildev` argument is not an open file descriptor.26628 [ENOTDIR] The open file descriptor `fildev` does not refer to a directory.26629 The `fchdir()` may fail if:26630 [EINTR] A signal was caught during the execution of `fchdir()`.

26631 [EIO] An I/O error occurred while reading from or writing to the file system.

26632 **EXAMPLES**

26633 None.

26634 **APPLICATION USAGE**

26635 None.

26636 **RATIONALE**

26637 None.

26638 **FUTURE DIRECTIONS**

26639 None.

26640 **SEE ALSO**26641 `chdir()`, `dirfd()`26642 XBD `<unistd.h>`26643 **CHANGE HISTORY**

26644 First released in Issue 4, Version 2.

26645 **Issue 5**

26646 Moved from X/OPEN UNIX extension to BASE.

26647 **Issue 7**26648 The `fchdir()` function is moved from the XSI option to the Base.

26649 **NAME**

26650 fchmod — change mode of a file

26651 **SYNOPSIS**

26652 #include &lt;sys/stat.h&gt;

26653 int fchmod(int *fildev*, mode\_t *mode*);26654 **DESCRIPTION**26655 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are  
26656 changed is specified by the file descriptor *fildev*.26657 SHM If *fildev* references a shared memory object, the *fchmod()* function need only affect the S\_IRUSR,  
26658 S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits.26659 TYM If *fildev* references a typed memory object, the behavior of *fchmod()* is unspecified.26660 If *fildev* refers to a socket, the behavior of *fchmod()* is unspecified.26661 OB XSR If *fildev* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call  
26662 returns successfully, doing nothing.26663 **RETURN VALUE**26664 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
26665 indicate the error.26666 **ERRORS**26667 The *fchmod()* function shall fail if:26668 [EBADF] The *fildev* argument is not an open file descriptor.26669 [EPERM] The effective user ID does not match the owner of the file and the process does  
26670 not have appropriate privileges.26671 [EROFS] The file referred to by *fildev* resides on a read-only file system.26672 The *fchmod()* function may fail if:26673 XSI [EINTR] The *fchmod()* function was interrupted by a signal.26674 XSI [EINVAL] The value of the *mode* argument is invalid.26675 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows  
26676 execution of *fchmod()* on a pipe.26677 **EXAMPLES**26678 **Changing the Current Permissions for a File**26679 The following example shows how to change the permissions for a file named */home/cnd/mod1*  
26680 so that the owner and group have read/write/execute permissions, but the world only has  
26681 read/write permissions.

26682 #include &lt;sys/stat.h&gt;

26683 #include &lt;fcntl.h&gt;

26684 mode\_t mode;

26685 int fildev;

26686 ...

26687 fildev = open("/home/cnd/mod1", O\_RDWR);

26688 fchmod(fildev, S\_IRWXU | S\_IRWXG | S\_IROTH | S\_IWOTH);

**fchmod()**

System Interfaces

26689 **APPLICATION USAGE**

26690 None.

26691 **RATIONALE**

26692 None.

26693 **FUTURE DIRECTIONS**

26694 None.

26695 **SEE ALSO**26696 *chmod(), chown(), creat(), fcntl(), fstatat(), fstatvfs(), mknod(), open(), read(), write()*26697 XBD [<sys/stat.h>](#)26698 **CHANGE HISTORY**

26699 First released in Issue 4, Version 2.

26700 **Issue 5**

26701 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX  
26702 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a  
26703 second instance of [EINVAL] is defined in the list of optional errors.

26704 **Issue 6**

26705 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*  
26706 behavior is unspecified for typed memory objects.

*System Interfaces***fchmodat()**26707 **NAME**

26708       fchmodat — change mode of a file relative to directory file descriptor

26709 **SYNOPSIS**

26710       #include &lt;sys/stat.h&gt;

26711       int fchmodat(int *fd*, const char \**path*, mode\_t *mode*, int *flag*);26712 **DESCRIPTION**26713       Refer to *chmod()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fchown()**26714 **NAME**26715 `fchown` — change owner and group of a file26716 **SYNOPSIS**26717 `#include <unistd.h>`26718 `int fchown(int fildes, uid_t owner, gid_t group);`26719 **DESCRIPTION**26720 The `fchown()` function shall be equivalent to `chown()` except that the file whose owner and group  
26721 are changed is specified by the file descriptor `filde`s.26722 **RETURN VALUE**26723 Upon successful completion, `fchown()` shall return 0. Otherwise, it shall return `-1` and set `errno` to  
26724 indicate the error.26725 **ERRORS**26726 The `fchown()` function shall fail if:26727 [EBADF] The `filde`s argument is not an open file descriptor.26728 [EPERM] The effective user ID does not match the owner of the file or the process does  
26729 not have appropriate privileges and `_POSIX_CHOWN_RESTRICTED`  
26730 indicates that such privilege is required.26731 [EROFS] The file referred to by `filde`s resides on a read-only file system.26732 The `fchown()` function may fail if:26733 [EINVAL] The owner or group ID is not a value supported by the implementation. The  
26734 OB XSR `filde`s argument refers to a pipe or socket or an `fattach()`-ed STREAM and the  
26735 implementation disallows execution of `fchown()` on a pipe.

26736 [EIO] A physical I/O error has occurred.

26737 [EINTR] The `fchown()` function was interrupted by a signal which was caught.26738 **EXAMPLES**26739 **Changing the Current Owner of a File**26740 The following example shows how to change the owner of a file named `/home/cnd/mod1` to  
26741 “jones” and the group to “cnd”.26742 The numeric value for the user ID is obtained by extracting the user ID from the user database  
26743 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by  
26744 extracting the group ID from the group database entry associated with “cnd”. This example  
26745 assumes the calling program has appropriate privileges.

```

26746 #include <sys/types.h>
26747 #include <unistd.h>
26748 #include <fcntl.h>
26749 #include <pwd.h>
26750 #include <grp.h>
26751
26751 struct passwd *pwd;
26752 struct group *grp;
26753 int
26754     filde;
26755
26755 ...
26755 filde = open("/home/cnd/mod1", O_RDWR);
26756 pwd = getpwnam("jones");

```

```
26757     grp = getgrnam("cnd");
26758     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

**26759 APPLICATION USAGE**

26760 None.

**26761 RATIONALE**

26762 None.

**26763 FUTURE DIRECTIONS**

26764 None.

**26765 SEE ALSO**

26766 [chown\(\)](#)

26767 XBD [<unistd.h>](#)

**26768 CHANGE HISTORY**

26769 First released in Issue 4, Version 2.

**26770 Issue 5**

26771 Moved from X/OPEN UNIX extension to BASE.

**26772 Issue 6**

26773 The following changes were made to align with the IEEE P1003.1a draft standard:

- 26774 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

26775 The *fchown()* function is defined as mandatory.

**26776 Issue 7**

26777 Functionality relating to XSI STREAMS is marked obsolescent.

**fchownat()***System Interfaces*26778 **NAME**26779 `fchownat` — change owner and group of a file relative to directory file descriptor26780 **SYNOPSIS**26781 `#include <unistd.h>`26782 `int fchownat(int fd, const char *path, uid_t owner, gid_t group,`26783 `int flag);`26784 **DESCRIPTION**26785 Refer to *chown()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

26786 **NAME**26787 `fclose` — close a stream26788 **SYNOPSIS**26789 `#include <stdio.h>`26790 `int fclose(FILE *stream);`26791 **DESCRIPTION**

26792 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26793 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26794 volume of POSIX.1-2008 defers to the ISO C standard.

26795 The `fclose()` function shall cause the stream pointed to by `stream` to be flushed and the associated  
 26796 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any  
 26797 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be  
 26798 disassociated from the file and any buffer set by the `setbuf()` or `setvbuf()` function shall be  
 26799 disassociated from the stream. If the associated buffer was automatically allocated, it shall be  
 26800 deallocated.

26801 CX If the file is not already at EOF, and the file is one capable of seeking, the file offset of the  
 26802 underlying open file description shall be adjusted so that the next operation on the open file  
 26803 description deals with the byte after the last one read from or written to the stream being closed.

26804 The `fclose()` function shall mark for update the last data modification and last file status change  
 26805 timestamps of the underlying file, if the stream was writable, and if buffered data remains that  
 26806 has not yet been written to the file. The `fclose()` function shall perform the equivalent of a `close()`  
 26807 on the file descriptor that is associated with the stream pointed to by `stream`.

26808 After the call to `fclose()`, any use of `stream` results in undefined behavior.

26809 **RETURN VALUE**

26810 CX Upon successful completion, `fclose()` shall return 0; otherwise, it shall return EOF and set `errno`  
 26811 to indicate the error.

26812 **ERRORS**

26813 The `fclose()` function shall fail if:

26814 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying `stream` and  
 26815 the thread would be delayed in the write operation.

26816 CX [EBADF] The file descriptor underlying stream is not valid.

26817 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

26818 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the  
 26819 process.

26820 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 26821 offset maximum associated with the corresponding stream.

26822 CX [EINTR] The `fclose()` function was interrupted by a signal.

26823 CX [EIO] The process is a member of a background process group attempting to write to  
 26824 its controlling terminal, TOSTOP is set, the process is neither ignoring nor  
 26825 blocking SIGTTOU, and the process group of the process is orphaned. This  
 26826 error may also be returned under implementation-defined conditions.

26827 CX [ENOMEM] The underlying stream was created by `open_memstream()` or  
 26828 `open_wmemstream()` and insufficient memory is available.

**fclose()**

26829 CX [ENOSPC] There was no free space remaining on the device containing the file or in the  
26830 buffer used by the *fmemopen()* function.

26831 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
26832 any process. A SIGPIPE signal shall also be sent to the thread.

26833 The *fclose()* function may fail if:

26834 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
26835 capabilities of the device.

**EXAMPLES**

26836 None.

**APPLICATION USAGE**

26838 None.

**RATIONALE**

26840 None.

**FUTURE DIRECTIONS**

26842 None.

**SEE ALSO**

26845 *close()*, *fmemopen()*, *fopen()*, *getrlimit()*, *open\_memstream()*, *ulimit()*

26846 XBD <stdio.h>

**CHANGE HISTORY**

26847 First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

26849 Large File Summit extensions are added.

**Issue 6**

26851 Extensions beyond the ISO C standard are marked.

26853 The following new requirements on POSIX implementations derive from alignment with the  
26854 Single UNIX Specification:

- 26855 • The [EFBIG] error is added as part of the large file support extensions.
- 26856 • The [ENXIO] optional error condition is added.

26857 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether  
26858 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

26859 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]  
26860 error in the ERRORS section from “the process would be delayed” to “the thread would be  
26861 delayed”.

**Issue 7**

26862 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file  
26863 descriptors and streams.

26865 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open  
26866 Group Technical Standard, 2006, Extended API Set Part 1.

26867 Changes are made related to support for finegrained timestamps.

26868 **NAME**

26869           fcntl — file control

26870 **SYNOPSIS**

26871           #include &lt;fcntl.h&gt;

26872           int fcntl(int *fildev*, int *cmd*, ...);26873 **DESCRIPTION**26874           The *fcntl()* function shall perform the operations described below on open files. The *fildev*  
26875           argument is a file descriptor.26876           The available values for *cmd* are defined in <fcntl.h> and are as follows:

26877           **F\_DUPFD**           Return a new file descriptor which shall be the lowest numbered  
26878           available (that is, not already open) file descriptor greater than or equal to  
26879           the third argument, *arg*, taken as an integer of type **int**. The new file  
26880           descriptor shall refer to the same open file description as the original file  
26881           descriptor, and shall share any locks. The FD\_CLOEXEC flag associated  
26882           with the new file descriptor shall be cleared to keep the file open across  
26883           calls to one of the *exec* functions.

26884           **F\_DUPFD\_CLOEXEC**26885           Like F\_DUPFD, but the FD\_CLOEXEC flag associated with the new file  
26886           descriptor shall be set.26887           **F\_GETFD**26888           Get the file descriptor flags defined in <fcntl.h> that are associated with  
26889           the file descriptor *fildev*. File descriptor flags are associated with a single  
26890           file descriptor and do not affect other file descriptors that refer to the  
26891           same file.26891           **F\_SETFD**26892           Set the file descriptor flags defined in <fcntl.h>, that are associated with  
26893           *fildev*, to the third argument, *arg*, taken as type **int**. If the FD\_CLOEXEC  
26894           flag in the third argument is 0, the file descriptor shall remain open across  
26895           the *exec* functions; otherwise, the file descriptor shall be closed upon  
26896           successful execution of one of the *exec* functions.26896           **F\_GETFL**26897           Get the file status flags and file access modes, defined in <fcntl.h>, for the  
26898           file description associated with *fildev*. The file access modes can be  
26899           extracted from the return value using the mask O\_ACCMODE, which is  
26900           defined in <fcntl.h>. File status flags and file access modes are associated  
26901           with the file description and do not affect other file descriptors that refer  
26902           to the same file with different open file descriptions. The flags returned  
26903           may include non-standard file status flags which the application did not  
26904           set, provided that these additional flags do not alter the behavior of a  
26905           conforming application.26905           **F\_SETFL**26906           Set the file status flags, defined in <fcntl.h>, for the file description  
26907           associated with *fildev* from the corresponding bits in the third argument,  
26908           *arg*, taken as type **int**. Bits corresponding to the file access mode and the  
26909           file creation flags, as defined in <fcntl.h>, that are set in *arg* shall be  
26910           ignored. If any bits in *arg* other than those mentioned here are changed by  
26911           the application, the result is unspecified.26911           **F\_GETOWN**26912           If *fildev* refers to a socket, get the process or process group ID specified to  
26913           receive SIGURG signals when out-of-band data is available. Positive  
26914           values indicate a process ID; negative values, other than -1, indicate a  
26915           process group ID. If *fildev* does not refer to a socket, the results are

**fcntl()**

|       |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26915 |          | unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 26916 | F_SETOWN | If <i>fildev</i> refers to a socket, set the process or process group ID specified to receive SIGURG signals when out-of-band data is available, using the value of the third argument, <i>arg</i> , taken as type <b>int</b> . Positive values indicate a process ID; negative values, other than $-1$ , indicate a process group ID. If <i>fildev</i> does not refer to a socket, the results are unspecified.                                                                                                                                                                                                                                                                               |
| 26917 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26918 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26919 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26920 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26921 |          | The following values for <i>cmd</i> are available for advisory record locking. Record locking shall be supported for regular files, and may be supported for other files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 26922 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26923 | F_GETLK  | Get the first lock which blocks the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type <b>struct flock</b> , defined in <b>&lt;fcntl.h&gt;</b> . The information retrieved shall overwrite the information passed to <i>fcntl()</i> in the structure <b>flock</b> . If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.                                                                                                                                                                                                            |
| 26924 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26925 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26926 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26927 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26928 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26929 | F_SETLK  | Set or clear a file segment lock according to the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type <b>struct flock</b> , defined in <b>&lt;fcntl.h&gt;</b> . F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in <b>&lt;fcntl.h&gt;</b> . If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall return immediately with a return value of $-1$ .                                                                                                                                            |
| 26930 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26931 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26932 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26933 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26934 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26935 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26936 | F_SETLKW | This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while <i>fcntl()</i> is waiting for a region, <i>fcntl()</i> shall be interrupted. Upon return from the signal handler, <i>fcntl()</i> shall return $-1$ with <i>errno</i> set to [EINTR], and the lock operation shall not be done.                                                                                                                                                                                                                                   |
| 26937 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26938 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26939 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26940 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26941 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26942 |          | Additional implementation-defined values for <i>cmd</i> may be defined in <b>&lt;fcntl.h&gt;</b> . Their names shall start with F_.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 26943 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26944 |          | When a shared lock is set on a segment of a file, other processes shall be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the file descriptor was not opened with read access.                                                                                                                                                                                                                                                                                                                                           |
| 26945 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26946 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26947 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26948 |          | An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock shall fail if the file descriptor was not opened with write access.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 26949 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26950 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26951 |          | The structure <b>flock</b> describes the type ( <i>l_type</i> ), starting offset ( <i>l_whence</i> ), relative offset ( <i>l_start</i> ), size ( <i>l_len</i> ), and process ID ( <i>l_pid</i> ) of the segment of the file to be affected.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 26952 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26953 |          | The value of <i>l_whence</i> is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative offset <i>l_start</i> bytes shall be measured from the start of the file, current position, or end of the file, respectively. The value of <i>l_len</i> is the number of consecutive bytes to be locked. The value of <i>l_len</i> may be negative (where the definition of <b>off_t</b> permits negative values of <i>l_len</i> ). The <i>l_pid</i> field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, when a blocking lock is found, the values returned in the <b>flock</b> structure shall be as follows: |
| 26954 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26955 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26956 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26957 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26958 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 26959 |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|       |                 |                                                                                                                                     |
|-------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| 26960 | <i>l_type</i>   | Type of blocking lock found.                                                                                                        |
| 26961 | <i>l_whence</i> | SEEK_SET.                                                                                                                           |
| 26962 | <i>l_start</i>  | Start of the blocking lock.                                                                                                         |
| 26963 | <i>l_len</i>    | Length of the blocking lock.                                                                                                        |
| 26964 | <i>l_pid</i>    | Process ID of the process that holds the blocking lock.                                                                             |
| 26965 |                 | If the command is F_SETLKW and the process must wait for another process to release a lock,                                         |
| 26966 |                 | then the range of bytes to be locked shall be determined before the <i>fcntl()</i> function blocks. If the                          |
| 26967 |                 | file size or file descriptor seek offset change while <i>fcntl()</i> is blocked, this shall not affect the                          |
| 26968 |                 | range of bytes locked.                                                                                                              |
| 26969 |                 | If <i>l_len</i> is positive, the area affected shall start at <i>l_start</i> and end at <i>l_start+l_len-1</i> . If <i>l_len</i> is |
| 26970 |                 | negative, the area affected shall start at <i>l_start+l_len</i> and end at <i>l_start-1</i> . Locks may start and                   |
| 26971 |                 | extend beyond the current end of a file, but shall not extend before the beginning of the file. A                                   |
| 26972 |                 | lock shall be set to extend to the largest possible value of the file offset for that file by setting                               |
| 26973 |                 | <i>l_len</i> to 0. If such a lock also has <i>l_start</i> set to 0 and <i>l_whence</i> is set to SEEK_SET, the whole file           |
| 26974 |                 | shall be locked.                                                                                                                    |
| 26975 |                 | There shall be at most one type of lock set for each byte in the file. Before a successful return                                   |
| 26976 |                 | from an F_SETLK or an F_SETLKW request when the calling process has previously existing                                             |
| 26977 |                 | locks on bytes in the region specified by the request, the previous lock type for each byte in the                                  |
| 26978 |                 | specified region shall be replaced by the new lock type. As specified above under the                                               |
| 26979 |                 | descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request                                                 |
| 26980 |                 | (respectively) shall fail or block when another process has existing locks on bytes in the specified                                |
| 26981 |                 | region and the type of any of those locks conflicts with the type specified in the request.                                         |
| 26982 |                 | All locks associated with a file for a given process shall be removed when a file descriptor for                                    |
| 26983 |                 | that file is closed by that process or the process holding that file descriptor terminates. Locks are                               |
| 26984 |                 | not inherited by a child process.                                                                                                   |
| 26985 |                 | A potential for deadlock occurs if a process controlling a locked region is put to sleep by                                         |
| 26986 |                 | attempting to lock the locked region of another process. If the system detects that sleeping until                                  |
| 26987 |                 | a locked region is unlocked would cause a deadlock, <i>fcntl()</i> shall fail with an [EDEADLK] error.                              |
| 26988 |                 | An unlock (F_UNLCK) request in which <i>l_len</i> is non-zero and the offset of the last byte of the                                |
| 26989 |                 | requested segment is the maximum value for an object of type <b>off_t</b> , when the process has an                                 |
| 26990 |                 | existing lock in which <i>l_len</i> is 0 and which includes the last byte of the requested segment, shall                           |
| 26991 |                 | be treated as a request to unlock from the start of the requested segment with an <i>l_len</i> equal to 0.                          |
| 26992 |                 | Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.                                          |
| 26993 | SHM             | When the file descriptor <i>fdes</i> refers to a shared memory object, the behavior of <i>fcntl()</i> shall be                      |
| 26994 |                 | the same as for a regular file except the effect of the following values for the argument <i>cmd</i> shall                          |
| 26995 |                 | be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.                                                                            |
| 26996 | TYM             | If <i>fdes</i> refers to a typed memory object, the result of the <i>fcntl()</i> function is unspecified.                           |
| 26997 |                 | <b>RETURN VALUE</b>                                                                                                                 |
| 26998 |                 | Upon successful completion, the value returned shall depend on <i>cmd</i> as follows:                                               |
| 26999 | F_DUPFD         | A new file descriptor.                                                                                                              |
| 27000 | F_DUPFD_CLOEXEC |                                                                                                                                     |
| 27001 |                 | A new file descriptor.                                                                                                              |

**fcntl()**

|       |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27002 | F_GETFD              | Value of flags defined in <code>&lt;fcntl.h&gt;</code> . The return value shall not be negative.                                                                                                                                                                                                                                                                                                                                                     |
| 27003 | F_SETFD              | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27004 | F_GETFL              | Value of file status flags and access modes. The return value is not negative.                                                                                                                                                                                                                                                                                                                                                                       |
| 27005 | F_SETFL              | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27006 | F_GETLK              | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27007 | F_SETLK              | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27008 | F_SETLKW             | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27009 | F_GETOWN             | Value of the socket owner process or process group; this will not be <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                               |
| 27010 | F_SETOWN             | Value other than <code>-1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 27011 |                      | Otherwise, <code>-1</code> shall be returned and <i>errno</i> set to indicate the error.                                                                                                                                                                                                                                                                                                                                                             |
| 27012 | <b>ERRORS</b>        |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27013 |                      | The <i>fcntl()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                           |
| 27014 | [EACCES] or [EAGAIN] |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27015 |                      | The <i>cmd</i> argument is <code>F_SETLK</code> ; the type of lock ( <i>l_type</i> ) is a shared ( <code>F_RDLCK</code> ) or exclusive ( <code>F_WRLCK</code> ) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.                                 |
| 27016 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27017 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27018 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27019 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27020 | [EBADF]              | The <i>fildev</i> argument is not a valid open file descriptor, or the argument <i>cmd</i> is <code>F_SETLK</code> or <code>F_SETLKW</code> , the type of lock, <i>l_type</i> , is a shared lock ( <code>F_RDLCK</code> ), and <i>fildev</i> is not a valid file descriptor open for reading, or the type of lock, <i>l_type</i> , is an exclusive lock ( <code>F_WRLCK</code> ), and <i>fildev</i> is not a valid file descriptor open for writing. |
| 27021 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27022 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27023 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27024 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27025 | [EINTR]              | The <i>cmd</i> argument is <code>F_SETLKW</code> and the function was interrupted by a signal.                                                                                                                                                                                                                                                                                                                                                       |
| 27026 | [EINVAL]             | The <i>cmd</i> argument is invalid, or the <i>cmd</i> argument is <code>F_DUPFD</code> or <code>F_DUPFD_CLOEXEC</code> and <i>arg</i> is negative or greater than or equal to <code>{OPEN_MAX}</code> , or the <i>cmd</i> argument is <code>F_GETLK</code> , <code>F_SETLK</code> , or <code>F_SETLKW</code> and the data pointed to by <i>arg</i> is not valid, or <i>fildev</i> refers to a file that does not support locking.                    |
| 27027 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27028 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27029 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27030 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27031 | [EMFILE]             | The argument <i>cmd</i> is <code>F_DUPFD</code> or <code>F_DUPFD_CLOEXEC</code> and all file descriptors available to the process are currently open, or no file descriptors greater than or equal to <i>arg</i> are available.                                                                                                                                                                                                                      |
| 27032 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27033 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27034 | [ENOLCK]             | The argument <i>cmd</i> is <code>F_SETLK</code> or <code>F_SETLKW</code> and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.                                                                                                                                                                                                                                      |
| 27035 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27036 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27037 | [EOVERFLOW]          | One of the values to be returned cannot be represented correctly.                                                                                                                                                                                                                                                                                                                                                                                    |
| 27038 | [EOVERFLOW]          | The <i>cmd</i> argument is <code>F_GETLK</code> , <code>F_SETLK</code> , or <code>F_SETLKW</code> and the smallest or, if <i>l_len</i> is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type <b>off_t</b> .                                                                                                                                                                      |
| 27039 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27040 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

27041 The *fcntl()* function may fail if:

27042 [EDEADLK] The *cmd* argument is F\_SETLKW, the lock is blocked by a lock from another  
 27043 process, and putting the calling process to sleep to wait for that lock to become  
 27044 free would cause a deadlock.

## 27045 EXAMPLES

### 27046 Locking and Unlocking a File

27047 The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then  
 27048 later remove it. F\_SETLK is used to perform a non-blocking lock request so that the process does  
 27049 not have to wait if an incompatible lock is held by another process; instead the process can take  
 27050 some other action.

```

27051 #include <stdlib.h>
27052 #include <unistd.h>
27053 #include <fcntl.h>
27054 #include <errno.h>
27055 #include <stdio.h>

27056 int
27057 main(int argc, char *argv[])
27058 {
27059     int fd;
27060     struct flock fl;

27061     fd = open("testfile", O_RDWR);
27062     if (fd == -1)
27063         /* Handle error */;

27064     /* Make a non-blocking request to place a write lock
27065        on bytes 100-109 of testfile */

27066     fl.l_type = F_WRLCK;
27067     fl.l_whence = SEEK_SET;
27068     fl.l_start = 100;
27069     fl.l_len = 10;

27070     if (fcntl(fd, F_SETLK, &fl) == -1) {
27071         if (errno == EACCES || errno == EAGAIN) {
27072             printf("Already locked by another process\n");
27073             /* We can't get the lock at the moment */
27074         } else {
27075             /* Handle unexpected error */;
27076         }
27077     } else { /* Lock was granted... */

27078         /* Perform I/O on bytes 100 to 109 of file */
27079         /* Unlock the locked bytes */

27080         fl.l_type = F_UNLCK;
27081         fl.l_whence = SEEK_SET;
27082         fl.l_start = 100;
27083         fl.l_len = 10;
27084         if (fcntl(fd, F_SETLK, &fl) == -1)
  
```

**fcntl()**

```

27085             /* Handle error */;
27086         }
27087         exit(EXIT_SUCCESS);
27088     } /* main */

```

### 27089 **Setting the Close-on-Exec Flag**

27090 The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```

27091 #include <unistd.h>
27092 #include <fcntl.h>
27093 ...
27094     int flags;
27095
27096     flags = fcntl(fd, F_GETFD);
27097     if (flags == -1)
27098         /* Handle error */;
27099     flags |= FD_CLOEXEC;
27100     if (fcntl(fd, F_SETFD, flags) == -1)
27101         /* Handle error */;

```

### 27101 **APPLICATION USAGE**

27102 The *arg* values to F\_GETFD, F\_SETFD, F\_GETFL, and F\_SETFL all represent flag values to allow  
 27103 for future growth. Applications using these functions should do a read-modify-write operation  
 27104 on them, rather than assuming that only the values defined by this volume of POSIX.1-2008 are  
 27105 valid. It is a common error to forget this, particularly in the case of F\_SETFD. Some  
 27106 implementations set additional file status flags to advise the application of default behavior,  
 27107 even though the application did not request these flags.

### 27108 **RATIONALE**

27109 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number  
 27110 of arguments. It is used because System V uses pointers for the implementation of file locking  
 27111 functions.

27112 This volume of POSIX.1-2008 permits concurrent read and write access to file data using the  
 27113 *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.  
 27114 Without concurrency controls, this feature may not be fully utilized without occasional loss of  
 27115 data.

27116 Data losses occur in several ways. One case occurs when several processes try to update the  
 27117 same record, without sequencing controls; several updates may occur in parallel and the last  
 27118 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing  
 27119 reorganization. Without exclusive use to the tree segment by the updating process, other reading  
 27120 processes chance getting lost in the database when the index blocks are split, condensed,  
 27121 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly  
 27122 general and does not handle the bit-tree example well.

27123 This facility is only required for regular files because it is not appropriate for many devices such  
 27124 as terminals and network connections.

27125 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,  
 27126 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may  
 27127 affect a file that another process also has open.

27128 The use of the open file description to identify what to lock requires extra calls and presents  
 27129 problems if several processes are sharing an open file description, but there are too many

27130 implementations of the existing mechanism for this volume of POSIX.1-2008 to use different  
27131 specifications.

27132 Another consequence of this model is that closing any file descriptor for a given file (whether or  
27133 not it is the same open file description that created the lock) causes the locks on that file to be  
27134 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the  
27135 locks owned on that file for that process. But note that while an open file description may be  
27136 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through  
27137 one of the *exec* functions.

27138 The identification of a machine in a network environment is outside the scope of this volume of  
27139 POSIX.1-2008. Thus, an *l\_sysid* member, such as found in System V, is not included in the locking  
27140 structure.

27141 Changing of lock types can result in a previously locked region being split into smaller regions.

27142 Mandatory locking was a major feature of the 1984 /usr/group standard.

27143 For advisory file record locking to be effective, all processes that have access to a file must  
27144 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode  
27145 record locking is important when it cannot be assumed that all processes are cooperating. For  
27146 example, if one user uses an editor to update a file at the same time that a second user executes  
27147 another process that updates the same file and if only one of the two processes is using advisory  
27148 locking, the processes are not cooperating. Enforcement-mode record locking would protect  
27149 against accidental collisions.

27150 Secondly, advisory record locking requires a process using locking to bracket each I/O operation  
27151 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a  
27152 process can lock the file once and unlock when all I/O operations have been completed.  
27153 Enforcement-mode record locking provides a base that can be enhanced; for example, with  
27154 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so  
27155 other processes could read it, but none of them could write it.

27156 Mandatory locks were omitted for several reasons:

- 27157 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most  
27158 implementations; this was confusing, at best.
- 27159 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
- 27160 3. Any publicly readable file could be locked by anyone. Many historical implementations  
27161 keep the password database in a publicly readable file. A malicious user could thus  
27162 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
- 27163 4. Some demand-paged historical implementations offer memory mapped files, and  
27164 enforcement cannot be done on that type of file.

27165 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a  
27166 timeout facility in applications requiring it. This is useful in deadlock detection. Since  
27167 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was  
27168 made optional.

#### 27169 FUTURE DIRECTIONS

27170 None.

**fcntl()**27171 **SEE ALSO**27172 *alarm()*, *close()*, *exec*, *open()*, *sigaction()*27173 XBD **<fcntl.h>**, **<signal.h>**27174 **CHANGE HISTORY**

27175 First released in Issue 1. Derived from Issue 1 of the SVID.

27176 **Issue 5**27177 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
27178 Threads Extension.

27179 Large File Summit extensions are added.

27180 **Issue 6**27181 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.27182 The following new requirements on POSIX implementations derive from alignment with the  
27183 Single UNIX Specification:27184 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
27185 required for conforming implementations of previous POSIX specifications, it was not  
27186 required for UNIX applications.27187 • In the DESCRIPTION, sentences describing behavior when *l\_len* is negative are now  
27188 mandated, and the description of unlock (F\_UNLOCK) when *l\_len* is non-negative is  
27189 mandated.27190 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd*  
27191 is invalid, and two [EOVERFLOW] error conditions are added.

27192 The F\_GETOWN and F\_SETOWN values are added for sockets.

27193 The following changes were made to align with the IEEE P1003.1a draft standard:

27194 • Clarification is added that the extent of the bytes locked is determined prior to the  
27195 blocking action.27196 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
27197 *fcntl()* results are unspecified for typed memory objects.

27198 The normative text is updated to avoid use of the term “must” for application requirements.

27199 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the  
27200 EXAMPLES section.27201 **Issue 7**27202 Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned  
27203 when *cmd* is F\_GETFL.27204 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
27205 FD\_CLOEXEC flag atomically at *open()*, and adding the F\_DUPFD\_CLOEXEC flag.27206 The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines  
27207 SEEK\_SET, SEEK\_CUR, and SEEK\_END as part of the Base.

27208 **NAME**27209 `fdatasync` — synchronize the data of a file (**REALTIME**)27210 **SYNOPSIS**

```
27211 SIO #include <unistd.h>
27212 int fdatasync(int fildev);
```

27213 **DESCRIPTION**

27214 The `fdatasync()` function shall force all currently queued I/O operations associated with the file  
 27215 indicated by file descriptor *fildev* to the synchronized I/O completion state.

27216 The functionality shall be equivalent to `fsync()` with the symbol `_POSIX_SYNCHRONIZED_IO`  
 27217 defined, with the exception that all I/O operations shall be completed as defined for  
 27218 synchronized I/O data integrity completion.

27219 **RETURN VALUE**

27220 If successful, the `fdatasync()` function shall return the value 0; otherwise, the function shall return  
 27221 the value `-1` and set `errno` to indicate the error. If the `fdatasync()` function fails, outstanding I/O  
 27222 operations are not guaranteed to have been completed.

27223 **ERRORS**

27224 The `fdatasync()` function shall fail if:

27225 [EBADF] The *fildev* argument is not a valid file descriptor open for writing.

27226 [EINVAL] This implementation does not support synchronized I/O for this file.

27227 In the event that any of the queued I/O operations fail, `fdatasync()` shall return the error  
 27228 conditions defined for `read()` and `write()`.

27229 **EXAMPLES**

27230 None.

27231 **APPLICATION USAGE**

27232 None.

27233 **RATIONALE**

27234 None.

27235 **FUTURE DIRECTIONS**

27236 None.

27237 **SEE ALSO**

27238 `aio_fsync()`, `fcntl()`, `fsync()`, `open()`, `read()`, `write()`

27239 XBD `<unistd.h>`

27240 **CHANGE HISTORY**

27241 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

27242 **Issue 6**

27243 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 27244 implementation does not support the Synchronized Input and Output option.

27245 The `fdatasync()` function is marked as part of the Synchronized Input and Output option.

**fdetach()**27246 **NAME**27247 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)27248 **SYNOPSIS**

```
27249 OB XSR #include <stropts.h>
27250 int fdetach(const char *path);
```

27251 **DESCRIPTION**

27252 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached  
 27253 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached  
 27254 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A  
 27255 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to  
 27256 again name the file to which the STREAMS file was attached. All subsequent operations on *path*  
 27257 shall operate on the underlying file and not on the STREAMS file.

27258 All open file descriptions established while the STREAMS file was attached to the file referenced  
 27259 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

27260 If there are no open file descriptors or other references to the STREAMS file, then a successful  
 27261 call to *fdetach()* shall be equivalent to performing the last *close()* on the attached file.

27262 **RETURN VALUE**

27263 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 27264 indicate the error.

27265 **ERRORS**

27266 The *fdetach()* function shall fail if:

- |       |                |                                                                                                                                                                                                                                                                                                                                    |
|-------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27267 | [EACCES]       | Search permission is denied on a component of the path prefix.                                                                                                                                                                                                                                                                     |
| 27268 | [EINVAL]       | The <i>path</i> argument names a file that is not currently attached.                                                                                                                                                                                                                                                              |
| 27269 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                                                                                                                                                                                                                                         |
| 27270 |                |                                                                                                                                                                                                                                                                                                                                    |
| 27271 | [ENAMETOOLONG] | The length of a component of a pathname is longer than {NAME_MAX}.                                                                                                                                                                                                                                                                 |
| 27272 |                |                                                                                                                                                                                                                                                                                                                                    |
| 27273 | [ENOENT]       | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.                                                                                                                                                                                                                                       |
| 27274 | [ENOTDIR]      | A component of the path prefix is not a directory, or the <i>path</i> argument contains at least one non- <i>&lt;slash&gt;</i> character and ends with one or more trailing <i>&lt;slash&gt;</i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory. |
| 27275 |                |                                                                                                                                                                                                                                                                                                                                    |
| 27276 |                |                                                                                                                                                                                                                                                                                                                                    |
| 27277 |                |                                                                                                                                                                                                                                                                                                                                    |
| 27278 | [EPERM]        | The effective user ID is not the owner of <i>path</i> and the process does not have appropriate privileges.                                                                                                                                                                                                                        |
| 27279 |                |                                                                                                                                                                                                                                                                                                                                    |

27280 The *fdetach()* function may fail if:

- |       |                |                                                                                                                                                               |
|-------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27281 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                                        |
| 27282 |                |                                                                                                                                                               |
| 27283 | [ENAMETOOLONG] | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}. |
| 27284 |                |                                                                                                                                                               |
| 27285 |                |                                                                                                                                                               |
| 27286 |                |                                                                                                                                                               |

27287 **EXAMPLES**27288 **Detaching a File**

27289 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to  
 27290 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this  
 27291 file refer to the underlying file, not to the STREAMS file.

```
27292 #include <stropts.h>
27293 ...
27294     char *filename = "/tmp/named-STREAM";
27295     int ret;
27296
27297     ret = fdetach(filename);
```

27297 **APPLICATION USAGE**

27298 None.

27299 **RATIONALE**

27300 None.

27301 **FUTURE DIRECTIONS**

27302 The *fdetach()* function may be removed in a future version.

27303 **SEE ALSO**

27304 *fattach()*

27305 XBD <*stropts.h*>

27306 **CHANGE HISTORY**

27307 First released in Issue 4, Version 2.

27308 **Issue 5**

27309 Moved from X/OPEN UNIX extension to BASE.

27310 **Issue 6**

27311 The normative text is updated to avoid use of the term “must” for application requirements.

27312 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 27313 [ELOOP] error condition is added.

27314 **Issue 7**

27315 Austin Group Interpretation 1003.1-2001 #143 is applied.

27316 The *fdetach()* function is marked obsolescent.

27317 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 27318 pathname exists but is not a directory or a symbolic link to a directory.

**fdim()**27319 **NAME**27320 `fdim`, `fdimf`, `fdiml` — compute positive difference between two floating-point numbers27321 **SYNOPSIS**

```
27322 #include <math.h>
27323 double fdim(double x, double y);
27324 float fdimf(float x, float y);
27325 long double fdiml(long double x, long double y);
```

27326 **DESCRIPTION**

27327 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27328 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27329 volume of POSIX.1-2008 defers to the ISO C standard.

27330 These functions shall determine the positive difference between their arguments. If  $x$  is greater  
 27331 than  $y$ ,  $x-y$  is returned. If  $x$  is less than or equal to  $y$ ,  $+0$  is returned.

27332 An application wishing to check for error situations should set `errno` to zero and call  
 27333 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 27334 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 27335 zero, an error has occurred.

27336 **RETURN VALUE**

27337 Upon successful completion, these functions shall return the positive difference value.

27338 If  $x-y$  is positive and overflows, a range error shall occur and `fdim()`, `fdimf()`, and `fdiml()` shall  
 27339 return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

27340 XSI If  $x-y$  is positive and underflows, a range error may occur, and either  $(x-y)$  (if representable), or  
 27341 `0.0` (if supported), or an implementation-defined value shall be returned.

27342 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

27343 **ERRORS**27344 The `fdim()` function shall fail if:

27345 Range Error The result overflows.

27346 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 27347 then `errno` shall be set to `[ERANGE]`. If the integer expression  
 27348 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow  
 27349 floating-point exception shall be raised.

27350 The `fdim()` function may fail if:

27351 Range Error The result underflows.

27352 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 27353 then `errno` shall be set to `[ERANGE]`. If the integer expression  
 27354 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow  
 27355 floating-point exception shall be raised.

27356 **EXAMPLES**

27357 None.

27358 **APPLICATION USAGE**

27359 On implementations supporting IEEE Std 754-1985,  $x-y$  cannot underflow, and hence the 0.0  
27360 return value is shaded as an extension for implementations supporting the XSI option rather  
27361 than an MX extension.

27362 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
27363 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

27364 **RATIONALE**

27365 None.

27366 **FUTURE DIRECTIONS**

27367 None.

27368 **SEE ALSO**27369 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [fmax\(\)](#), [fmin\(\)](#)27370 [Section 4.19](#) (on page 116), [<math.h>](#)27371 **CHANGE HISTORY**

27372 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**fdopen()**27373 **NAME**27374 `fdopen` — associate a stream with a file descriptor27375 **SYNOPSIS**

```
27376 CX #include <stdio.h>
27377 FILE *fdopen(int fildev, const char *mode);
```

27378 **DESCRIPTION**27379 The `fdopen()` function shall associate a stream with a file descriptor.27380 The *mode* argument is a character string having one of the following values:27381 *r* or *rb* Open a file for reading.27382 *w* or *wb* Open a file for writing.27383 *a* or *ab* Open a file for writing at end-of-file.27384 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).27385 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).27386 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end-of-file.27387 The meaning of these flags is exactly as specified in `fopen()`, except that modes beginning with *w*  
27388 shall not cause truncation of the file.27389 Additional values for the *mode* argument may be supported by an implementation.27390 The application shall ensure that the mode of the stream as expressed by the *mode* argument is  
27391 allowed by the file access mode of the open file description to which *fildev* refers. The file  
27392 position indicator associated with the new stream is set to the position indicated by the file offset  
27393 associated with the file descriptor.27394 The error and end-of-file indicators for the stream shall be cleared. The `fdopen()` function may  
27395 cause the last data access timestamp of the underlying file to be marked for update.27396 SHM If *fildev* refers to a shared memory object, the result of the `fdopen()` function is unspecified.27397 TYM If *fildev* refers to a typed memory object, the result of the `fdopen()` function is unspecified.27398 The `fdopen()` function shall preserve the offset maximum previously set for the open file  
27399 description corresponding to *fildev*.27400 **RETURN VALUE**27401 Upon successful completion, `fdopen()` shall return a pointer to a stream; otherwise, a null pointer  
27402 shall be returned and *errno* set to indicate the error.27403 **ERRORS**27404 The `fdopen()` function shall fail if:

27405 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

27406 The `fdopen()` function may fail if:27407 [EBADF] The *fildev* argument is not a valid file descriptor.27408 [EINVAL] The *mode* argument is not a valid mode.

27409 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

27410 [ENOMEM] Insufficient space to allocate a buffer.

#### 27411 EXAMPLES

27412 None.

#### 27413 APPLICATION USAGE

27414 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but  
27415 do not return streams.

#### 27416 RATIONALE

27417 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, *fcntl()*, or *socket()*;  
27418 inherited through *fork()*, *posix\_spawn()*, or *exec*; or perhaps obtained by other means.

27419 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write  
27420 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument  
27421 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.  
27422 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of  
27423 POSIX.1-2008, a good implementation of append (*a*) mode would cause the O\_APPEND flag to  
27424 be set.

#### 27425 FUTURE DIRECTIONS

27426 None.

#### 27427 SEE ALSO

27428 [Section 2.5.1](#) (on page 491), *fclose()*, *fmemopen()*, *fopen()*, *open()*, *open\_memstream()*,  
27429 *posix\_spawn()*, *socket()*

27430 XBD <stdio.h>

#### 27431 CHANGE HISTORY

27432 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 27433 Issue 5

27434 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

27435 Large File Summit extensions are added.

#### 27436 Issue 6

27437 The following new requirements on POSIX implementations derive from alignment with the  
27438 Single UNIX Specification:

- 27439 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include  
27440 binary streams.
- 27441 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset  
27442 maximum in the open file description.
- 27443 • All errors identified in the ERRORS section are added.
- 27444 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st\_atime* to be  
27445 updated.

27446 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27447 • Clarification is added that it is the responsibility of the application to ensure that the mode  
27448 is compatible with the open file descriptor.

27449 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
27450 *fdopen()* results are unspecified for typed memory objects.

**fdopen()**

- 27451 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the  
27452 RATIONALE.
- 27453 **Issue 7**
- 27454 SD5-XSH-ERN-149 is applied, adding the {STREAM\_MAX} [EMFILE] error condition.
- 27455 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

27456 **NAME**27457 `fdopendir, opendir` — open directory associated with file descriptor27458 **SYNOPSIS**

```
27459 #include <dirent.h>
27460 DIR *fdopendir(int fd);
27461 DIR *opendir(const char *dirname);
```

27462 **DESCRIPTION**

27463 The `fdopendir()` function shall be equivalent to the `opendir()` function except that the directory is  
 27464 specified by a file descriptor rather than by a name. The file offset associated with the file  
 27465 descriptor at the time of the call determines which entries are returned.

27466 Upon successful return from `fdopendir()`, the file descriptor is under the control of the system,  
 27467 and if any attempt is made to close the file descriptor, or to modify the state of the associated  
 27468 description, other than by means of `closedir()`, `readdir()`, `readdir_r()`, or `rewinddir()`, the behavior  
 27469 is undefined. Upon calling `closedir()` the file descriptor shall be closed.

27470 It is unspecified whether the `FD_CLOEXEC` flag will be set on the file descriptor by a successful  
 27471 call to `fdopendir()`.

27472 The `opendir()` function shall open a directory stream corresponding to the directory named by  
 27473 the `dirname` argument. The directory stream is positioned at the first entry. If the type **DIR**  
 27474 is implemented using a file descriptor, applications shall only be able to open up to a total of  
 27475 `{OPEN_MAX}` files and directories.

27476 If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the  
 27477 `O_DIRECTORY` flag was passed to `open()`.

27478 **RETURN VALUE**

27479 Upon successful completion, these functions shall return a pointer to an object of type **DIR**.  
 27480 Otherwise, these functions shall return a null pointer and set `errno` to indicate the error.

27481 **ERRORS**

27482 The `fdopendir()` function shall fail if:

27483 [EBADF] The `fd` argument is not a valid file descriptor open for reading.

27484 [ENOTDIR] The descriptor `fd` is not associated with a directory.

27485 The `opendir()` function shall fail if:

27486 [EACCES] Search permission is denied for the component of the path prefix of `dirname` or  
 27487 read permission is denied for `dirname`.

27488 [ELOOP] A loop exists in symbolic links encountered during resolution of the `dirname`  
 27489 argument.

27490 [ENAMETOOLONG] The length of a component of a pathname is longer than `{NAME_MAX}`.

27492 [ENOENT] A component of `dirname` does not name an existing directory or `dirname` is an  
 27493 empty string.

27494 [ENOTDIR] A component of `dirname` is not a directory.

**fdopendir()**

- 27495 The *opendir()* function may fail if:
- 27496 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
27497 resolution of the *dirname* argument.
- 27498 [EMFILE] All file descriptors available to the process are currently open.
- 27499 [ENAMETOOLONG]  
27500 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
27501 symbolic link produced an intermediate result with a length that exceeds  
27502 {PATH\_MAX}.
- 27503 [ENFILE] Too many files are currently open in the system.

27504 **EXAMPLES**27505 **Open a Directory Stream**

27506 The following program fragment demonstrates how the *opendir()* function is used.

```
27507 #include <dirent.h>
27508 ...
27509     DIR *dir;
27510     struct dirent *dp;
27511 ...
27512     if ((dir = opendir (".")) == NULL) {
27513         perror ("Cannot open .");
27514         exit (1);
27515     }
27516     while ((dp = readdir (dir)) != NULL) {
27517         ...
```

27518 **Find And Open a File**

27519 The following program searches through a given directory looking for files whose name does  
27520 not begin with a dot and whose size is larger than 1 MiB.

```
27521 #include <stdio.h>
27522 #include <dirent.h>
27523 #include <fcntl.h>
27524 #include <sys/stat.h>
27525 #include <stdint.h>
27526 #include <stdlib.h>
27527 #include <unistd.h>
27528 int
27529 main(int argc, char *argv[])
27530 {
27531     struct stat statbuf;
27532     DIR *d;
27533     struct dirent *dp;
27534     int dfd, ffd;
27535     if ((d = fdopendir((dfd = open("./tmp", O_RDONLY))) == NULL) {
27536         fprintf(stderr, "Cannot open ./tmp directory\n");
27537         exit(1);
```

```

27538     }
27539     while ((dp = readdir(d)) != NULL) {
27540         if (dp->d_name[0] == '.')
27541             continue;
27542         /* there is a possible race condition here as the file
27543          * could be renamed between the readdir and the open */
27544         if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
27545             perror(dp->d_name);
27546             continue;
27547         }
27548         if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
27549             /* found it ... */
27550             printf("%s: %jdK\n", dp->d_name,
27551                 (intmax_t)(statbuf.st_size / 1024));
27552         }
27553         close(ffd);
27554     }
27555     closedir(d); // note this implicitly closes dfd
27556     return 0;
27557 }

```

#### 27558 APPLICATION USAGE

27559 The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to  
 27560 examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is  
 27561 recommended for portability.

#### 27562 RATIONALE

27563 The purpose of the *fdopendir()* function is to enable opening files in directories other than the  
 27564 current working directory without exposure to race conditions. Any part of the path of a file  
 27565 could be changed in parallel to a call to *opendir()*, resulting in unspecified behavior.

27566 Based on historical implementations, the rules about file descriptors apply to directory streams  
 27567 as well. However, this volume of POSIX.1-2008 does not mandate that the directory stream be  
 27568 implemented using file descriptors. The description of *closedir()* clarifies that if a file descriptor  
 27569 is used for the directory stream, it is mandatory that *closedir()* deallocate the file descriptor.  
 27570 When a file descriptor is used to implement the directory stream, it behaves as if the  
 27571 FD\_CLOEXEC had been set for the file descriptor.

27572 The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2008 does not  
 27573 provide a way to test *a priori* for their existence because an application that is portable must be  
 27574 written to look for (and usually ignore) those entries. Writing code that presumes that they are  
 27575 the first two entries does not always work, as many implementations permit them to be other  
 27576 than the first two entries, with a “normal” entry preceding them. There is negligible value in  
 27577 providing a way to determine what the implementation does because the code to deal with dot  
 27578 and dot-dot must be written in any case and because such a flag would add to the list of those  
 27579 flags (which has proven in itself to be objectionable) and might be abused.

27580 Since the structure and buffer allocation, if any, for directory operations are defined by the  
 27581 implementation, this volume of POSIX.1-2008 imposes no portability requirements for erroneous  
 27582 program constructs, erroneous data, or the use of unspecified values such as the use or  
 27583 referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed or  
 27584 after a *fork()* or one of the *exec* function calls.

**fdopendir()**27585 **FUTURE DIRECTIONS**

27586 None.

27587 **SEE ALSO**27588 *closedir()*, *dirfd()*, *fstatat()*, *open()*, *readdir()*, *rewinddir()*, *symlink()*27589 XBD **<dirent.h>**, **<sys/types.h>**27590 **CHANGE HISTORY**

27591 First released in Issue 2.

27592 **Issue 6**27593 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.27594 The following new requirements on POSIX implementations derive from alignment with the  
27595 Single UNIX Specification:

- 27596 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
27597 required for conforming implementations of previous POSIX specifications, it was not  
27598 required for UNIX applications.
- 27599 • The [ELOOP] mandatory error condition is added.
- 27600 • A second [ENAMETOOLONG] is added as an optional error condition.

27601 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27602 • The [ELOOP] optional error condition is added.

27603 **Issue 7**

27604 Austin Group Interpretation 1003.1-2001 #143 is applied.

27605 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

27606 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API  
27607 Set Part 2.

27608 An additional example is added.

27609 **NAME**27610 `feclearexcept` — clear floating-point exception27611 **SYNOPSIS**27612 `#include <fenv.h>`27613 `int feclearexcept(int excepts);`27614 **DESCRIPTION**27615 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
27616 conflict between the requirements described here and the ISO C standard is unintentional. This  
27617 volume of POSIX.1-2008 defers to the ISO C standard.27618 The `feclearexcept()` function shall attempt to clear the supported floating-point exceptions  
27619 represented by *excepts*.27620 **RETURN VALUE**27621 If the argument is zero or if all the specified exceptions were successfully cleared, `feclearexcept()`  
27622 shall return zero. Otherwise, it shall return a non-zero value.27623 **ERRORS**

27624 No errors are defined.

27625 **EXAMPLES**

27626 None.

27627 **APPLICATION USAGE**

27628 None.

27629 **RATIONALE**

27630 None.

27631 **FUTURE DIRECTIONS**

27632 None.

27633 **SEE ALSO**27634 [fegetexceptflag\(\)](#), [feraiseexcept\(\)](#), [fetestexcept\(\)](#)27635 XBD [<fenv.h>](#)27636 **CHANGE HISTORY**

27637 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27638 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

**fegetenv()**27639 **NAME**

27640 fegetenv, fesetenv — get and set current floating-point environment

27641 **SYNOPSIS**

```
27642 #include <fenv.h>
27643 int fegetenv(fenv_t *envp);
27644 int fesetenv(const fenv_t *envp);
```

27645 **DESCRIPTION**

27646 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 27647 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27648 volume of POSIX.1-2008 defers to the ISO C standard.

27649 The *fegetenv()* function shall attempt to store the current floating-point environment in the object  
 27650 pointed to by *envp*.

27651 The *fesetenv()* function shall attempt to establish the floating-point environment represented by  
 27652 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to  
 27653 *fegetenv()* or *fehldexcept()*, or equal a floating-point environment macro. The *fesetenv()* function  
 27654 does not raise floating-point exceptions, but only installs the state of the floating-point status  
 27655 flags represented through its argument.

27656 **RETURN VALUE**

27657 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall  
 27658 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return  
 27659 zero. Otherwise, it shall return a non-zero value.

27660 **ERRORS**

27661 No errors are defined.

27662 **EXAMPLES**

27663 None.

27664 **APPLICATION USAGE**

27665 None.

27666 **RATIONALE**

27667 None.

27668 **FUTURE DIRECTIONS**

27669 None.

27670 **SEE ALSO**27671 *fehldexcept()*, *feupdateenv()*

27672 XBD &lt;fenv.h&gt;

27673 **CHANGE HISTORY**

27674 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27675 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27676 **NAME**

27677 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

27678 **SYNOPSIS**

27679 #include &lt;fenv.h&gt;

27680 int fegetexceptflag(fexcept\_t \*flagp, int excepts);

27681 int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

27682 **DESCRIPTION**

27683 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 27684 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27685 volume of POSIX.1-2008 defers to the ISO C standard.

27686 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of  
 27687 the states of the floating-point status flags indicated by the argument *excepts* in the object  
 27688 pointed to by the argument *flagp*.

27689 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the  
 27690 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by  
 27691 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument  
 27692 represented at least those floating-point exceptions represented by the argument *excepts*. This  
 27693 function does not raise floating-point exceptions, but only sets the state of the flags.

27694 **RETURN VALUE**

27695 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it  
 27696 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions  
 27697 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero  
 27698 value.

27699 **ERRORS**

27700 No errors are defined.

27701 **EXAMPLES**

27702 None.

27703 **APPLICATION USAGE**

27704 None.

27705 **RATIONALE**

27706 None.

27707 **FUTURE DIRECTIONS**

27708 None.

27709 **SEE ALSO**27710 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*

27711 XBD &lt;fenv.h&gt;

27712 **CHANGE HISTORY**

27713 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27714 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

**fegetround()**27715 **NAME**

27716 fegetround, fesetround — get and set current rounding direction

27717 **SYNOPSIS**

```
27718 #include <fenv.h>
27719 int fegetround(void);
27720 int fesetround(int round);
```

27721 **DESCRIPTION**

27722 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27723 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27724 volume of POSIX.1-2008 defers to the ISO C standard.

27725 The *fegetround()* function shall get the current rounding direction.

27726 The *fesetround()* function shall establish the rounding direction represented by its argument  
 27727 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding  
 27728 direction is not changed.

27729 **RETURN VALUE**

27730 The *fegetround()* function shall return the value of the rounding direction macro representing the  
 27731 current rounding direction or a negative value if there is no such rounding direction macro or  
 27732 the current rounding direction is not determinable.

27733 The *fesetround()* function shall return a zero value if and only if the requested rounding direction  
 27734 was established.

27735 **ERRORS**

27736 No errors are defined.

27737 **EXAMPLES**

27738 The following example saves, sets, and restores the rounding direction, reporting an error and  
 27739 aborting if setting the rounding direction fails:

```
27740 #include <fenv.h>
27741 #include <assert.h>
27742 void f(int round_dir)
27743 {
27744     #pragma STDC FENV_ACCESS ON
27745     int save_round;
27746     int setround_ok;
27747     save_round = fegetround();
27748     setround_ok = fesetround(round_dir);
27749     assert(setround_ok == 0);
27750     /* ... */
27751     fesetround(save_round);
27752     /* ... */
27753 }
```

27754 **APPLICATION USAGE**

27755 None.

27756 **RATIONALE**

27757 None.

27758 **FUTURE DIRECTIONS**

27759 None.

27760 **SEE ALSO**27761 XBD [<fenv.h>](#)27762 **CHANGE HISTORY**

27763 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27764 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**feholdexcept()**27765 **NAME**

27766 feholdexcept — save current floating-point environment

27767 **SYNOPSIS**

27768 #include &lt;fenv.h&gt;

27769 int feholdexcept(fenv\_t \*envp);

27770 **DESCRIPTION**27771 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
27772 conflict between the requirements described here and the ISO C standard is unintentional. This  
27773 volume of POSIX.1-2008 defers to the ISO C standard.27774 The *feholdexcept()* function shall save the current floating-point environment in the object  
27775 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on  
27776 floating-point exceptions) mode, if available, for all floating-point exceptions.27777 **RETURN VALUE**27778 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception  
27779 handling was successfully installed.27780 **ERRORS**

27781 No errors are defined.

27782 **EXAMPLES**

27783 None.

27784 **APPLICATION USAGE**

27785 None.

27786 **RATIONALE**27787 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard  
27788 implementations which have the default non-stop mode and at least one other mode for trap  
27789 handling or aborting. If the implementation provides only the non-stop mode, then installing the  
27790 non-stop mode is trivial.27791 **FUTURE DIRECTIONS**

27792 None.

27793 **SEE ALSO**27794 [fegetenv\(\)](#), [feupdateenv\(\)](#)27795 XBD [<fenv.h>](#)27796 **CHANGE HISTORY**

27797 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27798 **NAME**

27799 feof — test end-of-file indicator on a stream

27800 **SYNOPSIS**

27801 #include &lt;stdio.h&gt;

27802 int feof(FILE \*stream);

27803 **DESCRIPTION**

27804 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27805 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27806 volume of POSIX.1-2008 defers to the ISO C standard.

27807 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.27808 **RETURN VALUE**27809 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.27810 **ERRORS**

27811 No errors are defined.

27812 **EXAMPLES**

27813 None.

27814 **APPLICATION USAGE**

27815 None.

27816 **RATIONALE**

27817 None.

27818 **FUTURE DIRECTIONS**

27819 None.

27820 **SEE ALSO**27821 *clearerr()*, *ferror()*, *fopen()*

27822 XBD &lt;stdio.h&gt;

27823 **CHANGE HISTORY**

27824 First released in Issue 1. Derived from Issue 1 of the SVID.

**feraiseexcept()**27825 **NAME**27826 `feraiseexcept` — raise floating-point exception27827 **SYNOPSIS**

```
27828 #include <fenv.h>
27829 int feraiseexcept(int excepts);
```

27830 **DESCRIPTION**

27831 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27832 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27833 volume of POSIX.1-2008 defers to the ISO C standard.

27834 The *feraiseexcept()* function shall attempt to raise the supported floating-point exceptions  
 27835 represented by the argument *excepts*. The order in which these floating-point exceptions are  
 27836 raised is unspecified. Whether the *feraiseexcept()* function additionally raises the inexact floating-  
 27837 point exception whenever it raises the overflow or underflow floating-point exception is  
 27838 implementation-defined.

27839 **RETURN VALUE**

27840 If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept()*  
 27841 shall return zero. Otherwise, it shall return a non-zero value.

27842 **ERRORS**

27843 No errors are defined.

27844 **EXAMPLES**

27845 None.

27846 **APPLICATION USAGE**

27847 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic  
 27848 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

27849 **RATIONALE**

27850 Raising overflow or underflow is allowed to also raise inexact because on some architectures the  
 27851 only practical way to raise an exception is to execute an instruction that has the exception as a  
 27852 side-effect. The function is not restricted to accept only valid coincident expressions for atomic  
 27853 operations, so the function can be used to raise exceptions accrued over several operations.

27854 **FUTURE DIRECTIONS**

27855 None.

27856 **SEE ALSO**27857 *feclearexcept()*, *fegetexceptflag()*, *fetestexcept()*27858 XBD `<fenv.h>`27859 **CHANGE HISTORY**

27860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27861 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27862 **NAME**27863 `ferror` — test error indicator on a stream27864 **SYNOPSIS**27865 `#include <stdio.h>`27866 `int ferror(FILE *stream);`27867 **DESCRIPTION**27868 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
27869 conflict between the requirements described here and the ISO C standard is unintentional. This  
27870 volume of POSIX.1-2008 defers to the ISO C standard.27871 The `ferror()` function shall test the error indicator for the stream pointed to by `stream`.27872 **RETURN VALUE**27873 The `ferror()` function shall return non-zero if and only if the error indicator is set for `stream`.27874 **ERRORS**

27875 No errors are defined.

27876 **EXAMPLES**

27877 None.

27878 **APPLICATION USAGE**

27879 None.

27880 **RATIONALE**

27881 None.

27882 **FUTURE DIRECTIONS**

27883 None.

27884 **SEE ALSO**27885 `clearerr()`, `feof()`, `fopen()`27886 XBD `<stdio.h>`27887 **CHANGE HISTORY**

27888 First released in Issue 1. Derived from Issue 1 of the SVID.

**fesetenv()***System Interfaces*27889 **NAME**

27890 fesetenv — set current floating-point environment

27891 **SYNOPSIS**

27892 #include &lt;fenv.h&gt;

27893 int fesetenv(const fenv\_t \*envp);

27894 **DESCRIPTION**27895 Refer to *fegetenv()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***fesetexceptflag()**27896 **NAME**

27897       fesetexceptflag — set floating-point status flags

27898 **SYNOPSIS**

27899       #include &lt;fenv.h&gt;

27900       int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

27901 **DESCRIPTION**27902       Refer to *fegetexceptflag()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fesetround()***System Interfaces*27903 **NAME**

27904           fesetround — set current rounding direction

27905 **SYNOPSIS**

27906           #include &lt;fenv.h&gt;

27907           int fesetround(int *round*);27908 **DESCRIPTION**27909           Refer to *fegetround()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

27910 **NAME**27911 `fetestexcept` — test floating-point exception flags27912 **SYNOPSIS**27913 `#include <fenv.h>`27914 `int fetestexcept(int excepts);`27915 **DESCRIPTION**27916 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
27917 conflict between the requirements described here and the ISO C standard is unintentional. This  
27918 volume of POSIX.1-2008 defers to the ISO C standard.27919 The `fetestexcept()` function shall determine which of a specified subset of the floating-point  
27920 exception flags are currently set. The `excepts` argument specifies the floating-point status flags to  
27921 be queried.27922 **RETURN VALUE**27923 The `fetestexcept()` function shall return the value of the bitwise-inclusive OR of the floating-point  
27924 exception macros corresponding to the currently set floating-point exceptions included in  
27925 `excepts`.27926 **ERRORS**

27927 No errors are defined.

27928 **EXAMPLES**27929 The following example calls function `f()` if an invalid exception is set, and then function `g()` if an  
27930 overflow exception is set:27931 `#include <fenv.h>`  
27932 `/* ... */`  
27933 `{`  
27934  `#pragma STDC FENV_ACCESS ON`  
27935  `int set_excepts;`  
27936  `feclearexcept(FE_INVALID | FE_OVERFLOW);`  
27937  `// maybe raise exceptions`  
27938  `set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);`  
27939  `if (set_excepts & FE_INVALID) f();`  
27940  `if (set_excepts & FE_OVERFLOW) g();`  
27941  `/* ... */`  
27942 `}`27943 **APPLICATION USAGE**

27944 None.

27945 **RATIONALE**

27946 None.

27947 **FUTURE DIRECTIONS**

27948 None.

27949 **SEE ALSO**27950 `feclearexcept()`, `fegetexceptflag()`, `feraiseexcept()`27951 XBD `<fenv.h>`

## **fetestexcept()**

27952 **CHANGE HISTORY**

27953 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

27954 **NAME**

27955 feupdateenv — update floating-point environment

27956 **SYNOPSIS**

```
27957 #include <fenv.h>
27958 int feupdateenv(const fenv_t *envp);
```

27959 **DESCRIPTION**

27960 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27961 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27962 volume of POSIX.1-2008 defers to the ISO C standard.

27963 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in  
 27964 its automatic storage, attempt to install the floating-point environment represented by the object  
 27965 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument  
 27966 *envp* shall point to an object set by a call to *fehldexcept()* or *fegetenv()*, or equal a floating-point  
 27967 environment macro.

27968 **RETURN VALUE**

27969 The *feupdateenv()* function shall return a zero value if and only if all the required actions were  
 27970 successfully carried out.

27971 **ERRORS**

27972 No errors are defined.

27973 **EXAMPLES**

27974 The following example shows sample code to hide spurious underflow floating-point  
 27975 exceptions:

```
27976 #include <fenv.h>
27977 double f(double x)
27978 {
27979     #pragma STDC FENV_ACCESS ON
27980     double result;
27981     fenv_t save_env;
27982     fehldexcept(&save_env);
27983     // compute result
27984     if (/* test spurious underflow */)
27985         feclearexcept(FE_UNDERFLOW);
27986     feupdateenv(&save_env);
27987     return result;
27988 }
```

27989 **APPLICATION USAGE**

27990 None.

27991 **RATIONALE**

27992 None.

27993 **FUTURE DIRECTIONS**

27994 None.

27995 **SEE ALSO**

27996 *fegetenv()*, *fehldexcept()*

27997 XBD <fenv.h>

## feupdateenv()

### 27998 **CHANGE HISTORY**

27999 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28000 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***fexecve**28001 **NAME**

28002       fexecve — execute a file

28003 **SYNOPSIS**

28004       #include &lt;unistd.h&gt;

28005       int fexecve(int *fd*, char \*const *argv*[], char \*const *envp*[]);28006 **DESCRIPTION**28007       Refer to *exec*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fflush()**28008 **NAME**

28009 fflush — flush a stream

28010 **SYNOPSIS**

28011 #include &lt;stdio.h&gt;

28012 int fflush(FILE \*stream);

28013 **DESCRIPTION**

28014 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 28015 conflict between the requirements described here and the ISO C standard is unintentional. This  
 28016 volume of POSIX.1-2008 defers to the ISO C standard.

28017 If *stream* points to an output stream or an update stream in which the most recent operation was  
 28018 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the  
 28019 last data modification and last file status change timestamps of the underlying file shall be  
 28020 marked for update.

28021 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the  
 28022 behavior is defined above.

28023 CX For a stream open for reading, if the file is not already at EOF, and the file is one capable of  
 28024 seeking, the file offset of the underlying open file description shall be adjusted so that the next  
 28025 operation on the open file description deals with the byte after the last one read from or written  
 28026 to the stream being flushed.

28027 **RETURN VALUE**

28028 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for  
 28029 CX the stream, return EOF, and set *errno* to indicate the error.

28030 **ERRORS**28031 The *fflush()* function shall fail if:

28032 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
 28033 the thread would be delayed in the write operation.

28034 CX [EBADF] The file descriptor underlying *stream* is not valid.

28035 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

28036 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the  
 28037 process.

28038 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 28039 offset maximum associated with the corresponding stream.

28040 CX [EINTR] The *fflush()* function was interrupted by a signal.

28041 CX [EIO] The process is a member of a background process group attempting to write to  
 28042 its controlling terminal, TOSTOP is set, the process is neither ignoring nor  
 28043 blocking SIGTTOU, and the process group of the process is orphaned. This  
 28044 error may also be returned under implementation-defined conditions.

28045 CX [ENOMEM] The underlying stream was created by *open\_memstream()* or  
 28046 *open\_wmemstream()* and insufficient memory is available.

28047 CX [ENOSPC] There was no free space remaining on the device containing the file or in the  
 28048 buffer used by the *fmemopen()* function.

28049 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
28050 any process. A SIGPIPE signal shall also be sent to the thread.

28051 The *fflush()* function may fail if:

28052 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
28053 capabilities of the device.

## 28054 EXAMPLES

### 28055 Sending Prompts to Standard Output

28056 The following example uses *printf()* calls to print a series of prompts for information the user  
28057 must enter from standard input. The *fflush()* calls force the output to standard output. The  
28058 *fflush()* function is used because standard output is usually buffered and the prompt may not  
28059 immediately be printed on the output or terminal. The *getline()* function calls read strings from  
28060 standard input and place the results in variables, for use later in the program.

```
28061 char *user;
28062 char *oldpasswd;
28063 char *newpasswd;
28064 ssize_t llen;
28065 size_t blen;
28066 struct termios term;
28067 tcflag_t saveflag;

28068 printf("User name: ");
28069 fflush(stdout);
28070 blen = 0;
28071 llen = getline(&user, &blen, stdin);
28072 user[llen-1] = 0;
28073 tcgetattr(fileno(stdin), &term);
28074 saveflag = term.c_lflag;
28075 term.c_lflag &= ~ECHO;
28076 tcsetattr(fileno(stdin), TCSANOW, &term);
28077 printf("Old password: ");
28078 fflush(stdout);
28079 blen = 0;
28080 llen = getline(&oldpasswd, &blen, stdin);
28081 oldpasswd[llen-1] = 0;

28082 printf("\nNew password: ");
28083 fflush(stdout);
28084 blen = 0;
28085 llen = getline(&newpasswd, &blen, stdin);
28086 newpasswd[llen-1] = 0;
28087 term.c_lflag = saveflag;
28088 tcsetattr(fileno(stdin), TCSANOW, &term);
28089 free(user);
28090 free(oldpasswd);
28091 free(newpasswd);
```

**fflush()**28092 **APPLICATION USAGE**

28093 None.

28094 **RATIONALE**

28095 Data buffered by the system may make determining the validity of the position of the current  
 28096 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*  
 28097 on streams open for *read()* is not mandated by POSIX.1-2008.

28098 **FUTURE DIRECTIONS**

28099 None.

28100 **SEE ALSO**28101 *fnmemopen()*, *getrlimit()*, *open\_memstream()*, *ulimit()*28102 XBD `<stdio.h>`28103 **CHANGE HISTORY**

28104 First released in Issue 1. Derived from Issue 1 of the SVID.

28105 **Issue 5**

28106 Large File Summit extensions are added.

28107 **Issue 6**

28108 Extensions beyond the ISO C standard are marked.

28109 The following new requirements on POSIX implementations derive from alignment with the  
 28110 Single UNIX Specification:

- 28111 • The [EFBIG] error is added as part of the large file support extensions.
- 28112 • The [ENXIO] optional error condition is added.

28113 The RETURN VALUE section is updated to note that the error indicator shall be set for the  
 28114 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

28115 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN]  
 28116 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 28117 delayed”.

28118 **Issue 7**

28119 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file  
 28120 descriptors and streams.

28121 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open  
 28122 Group Technical Standard, 2006, Extended API Set Part 1.

28123 The EXAMPLES section is revised.

28124 Changes are made related to support for finegrained timestamps.

28125 **NAME**

28126 ffs — find first set bit

28127 **SYNOPSIS**

```
28128 XSI #include <strings.h>
28129 int ffs(int i);
```

28130 **DESCRIPTION**

28131 The *ffs()* function shall find the first bit set (beginning with the least significant bit) in *i*, and  
 28132 return the index of that bit. Bits are numbered starting at one (the least significant bit).

28133 **RETURN VALUE**

28134 The *ffs()* function shall return the index of the first bit set. If *i* is 0, then *ffs()* shall return 0.

28135 **ERRORS**

28136 No errors are defined.

28137 **EXAMPLES**

28138 None.

28139 **APPLICATION USAGE**

28140 None.

28141 **RATIONALE**

28142 None.

28143 **FUTURE DIRECTIONS**

28144 None.

28145 **SEE ALSO**

28146 XBD [<strings.h>](#)

28147 **CHANGE HISTORY**

28148 First released in Issue 4, Version 2.

28149 **Issue 5**

28150 Moved from X/OPEN UNIX extension to BASE.

**fgetc()**28151 **NAME**28152 `fgetc` — get a byte from a stream28153 **SYNOPSIS**28154 `#include <stdio.h>`28155 `int fgetc(FILE *stream);`28156 **DESCRIPTION**28157 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
28158 conflict between the requirements described here and the ISO C standard is unintentional. This  
28159 volume of POSIX.1-2008 defers to the ISO C standard.28160 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is  
28161 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,  
28162 from the input stream pointed to by *stream*, and advance the associated file position indicator for  
28163 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple  
28164 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.28165 CX The *fgetc()* function may mark the last data access timestamp of the file associated with *stream*  
28166 for update. The last data access timestamp shall be marked for update by the first successful  
28167 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or  
28168 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.28169 **RETURN VALUE**28170 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to  
28171 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the  
28172 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,  
28173 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to  
28174 indicate the error.28175 **ERRORS**28176 The *fgetc()* function shall fail if data needs to be read and:28177 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying *stream* and  
28178 the thread would be delayed in the *fgetc()* operation.28179 CX **[EBADF]** The file descriptor underlying *stream* is not a valid file descriptor open for  
28180 reading.28181 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data  
28182 was transferred.28183 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process  
28184 group attempting to read from its controlling terminal, and either the process  
28185 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.  
28186 This error may also be generated for implementation-defined reasons.28187 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the  
28188 offset maximum associated with the corresponding stream.28189 The *fgetc()* function may fail if:28190 CX **[ENOMEM]** Insufficient storage space is available.28191 CX **[ENXIO]** A request was made of a nonexistent device, or the request was outside the  
28192 capabilities of the device.

28193 **EXAMPLES**

28194 None.

28195 **APPLICATION USAGE**

28196 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared  
 28197 against the integer constant EOF, the comparison may never succeed, because sign-extension of  
 28198 a variable of type **char** on widening to integer is implementation-defined.

28199 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
 28200 end-of-file condition.

28201 **RATIONALE**

28202 None.

28203 **FUTURE DIRECTIONS**

28204 None.

28205 **SEE ALSO**28206 *feof()*, *ferror()*, *fgets()*, *fread()*, *fscanf()*, *getchar()*, *getc()*, *gets()*, *ungetc()*

28207 XBD &lt;stdio.h&gt;

28208 **CHANGE HISTORY**

28209 First released in Issue 1. Derived from Issue 1 of the SVID.

28210 **Issue 5**

28211 Large File Summit extensions are added.

28212 **Issue 6**

28213 Extensions beyond the ISO C standard are marked.

28214 The following new requirements on POSIX implementations derive from alignment with the  
 28215 Single UNIX Specification:

- 28216 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 28217 • The [ENOMEM] and [ENXIO] optional error conditions are added.

28218 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 28219 • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the  
 28220 input stream is not set.
- 28221 • The RETURN VALUE section is updated to note that the error indicator shall be set for the  
 28222 stream.

28223 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN]  
 28224 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 28225 delayed”.

28226 **Issue 7**

28227 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark  
 28228 the last data access timestamp for update.

**fgetpos()**28229 **NAME**28230 `fgetpos` — get current file position information28231 **SYNOPSIS**28232 `#include <stdio.h>`28233 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`28234 **DESCRIPTION**28235 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
28236 conflict between the requirements described here and the ISO C standard is unintentional. This  
28237 volume of POSIX.1-2008 defers to the ISO C standard.28238 The `fgetpos()` function shall store the current values of the parse state (if any) and file position  
28239 indicator for the stream pointed to by `stream` in the object pointed to by `pos`. The value stored  
28240 contains unspecified information usable by `fsetpos()` for repositioning the stream to its position  
28241 at the time of the call to `fgetpos()`.28242 **RETURN VALUE**28243 Upon successful completion, `fgetpos()` shall return 0; otherwise, it shall return a non-zero value  
28244 and set `errno` to indicate the error.28245 **ERRORS**28246 The `fgetpos()` function shall fail if:28247 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an  
28248 object of type `fpos_t`.28249 The `fgetpos()` function may fail if:28250 CX [EBADF] The file descriptor underlying `stream` is not valid.28251 CX [ESPIPE] The file descriptor underlying `stream` is associated with a pipe, FIFO, or socket.28252 **EXAMPLES**

28253 None.

28254 **APPLICATION USAGE**

28255 None.

28256 **RATIONALE**

28257 None.

28258 **FUTURE DIRECTIONS**

28259 None.

28260 **SEE ALSO**28261 `fopen()`, `ftell()`, `rewind()`, `ungetc()`28262 XBD `<stdio.h>`28263 **CHANGE HISTORY**

28264 First released in Issue 4. Derived from the ISO C standard.

28265 **Issue 5**

28266 Large File Summit extensions are added.

28267 **Issue 6**

28268 Extensions beyond the ISO C standard are marked.

28269 The following new requirements on POSIX implementations derive from alignment with the  
28270 Single UNIX Specification:

28271 • The [EBADF] and [ESPIPE] optional error conditions are added.

28272 An additional [ESPIPE] error condition is added for sockets.

28273 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fgets()**28274 **NAME**

28275           fgets — get a string from a stream

28276 **SYNOPSIS**

28277           #include &lt;stdio.h&gt;

28278           char \*fgets(char \*restrict *s*, int *n*, FILE \*restrict *stream*);28279 **DESCRIPTION**28280 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
28281 conflict between the requirements described here and the ISO C standard is unintentional. This  
28282 volume of POSIX.1-2008 defers to the ISO C standard.28283       The *fgets()* function shall read bytes from *stream* into the array pointed to by *s*, until *n*–1 bytes  
28284 are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.  
28285 The string is then terminated with a null byte.28286 CX       The *fgets()* function may mark the last data access timestamp of the file associated with *stream*  
28287 for update. The last data access timestamp shall be marked for update by the first successful  
28288 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or  
28289 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.28290 **RETURN VALUE**28291       Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file  
28292 indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,  
28293 CX       the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set  
28294 *errno* to indicate the error.28295 **ERRORS**28296       Refer to *fgetc()*.28297 **EXAMPLES**28298       **Reading Input**28299       The following example uses *fgets()* to read each line of input. {LINE\_MAX}, which defines the  
28300 maximum size of the input line, is defined in the <limits.h> header.28301       #include <stdio.h>  
28302       ...  
28303       char line[LINE\_MAX];  
28304       ...  
28305       while (fgets(line, LINE\_MAX, fp) != NULL) {  
28306       ...  
28307       }  
28308       .28309 **APPLICATION USAGE**

28310       None.

28311 **RATIONALE**

28312       None.

28313 **FUTURE DIRECTIONS**

28314       None.

28315 **SEE ALSO**28316 *fggetc()*, *fopen()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *gets()*, *ungetc()*

28317 XBD &lt;stdio.h&gt;

28318 **CHANGE HISTORY**

28319 First released in Issue 1. Derived from Issue 1 of the SVID.

28320 **Issue 6**

28321 Extensions beyond the ISO C standard are marked.

28322 The prototype for *fgets()* is changed for alignment with the ISO/IEC 9899:1999 standard.28323 **Issue 7**28324 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark  
28325 the last data access timestamp for update.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fgetwc()**28326 **NAME**28327 `fgetwc` — get a wide-character code from a stream28328 **SYNOPSIS**28329 `#include <stdio.h>`28330 `#include <wchar.h>`28331 `wint_t fgetwc(FILE *stream);`28332 **DESCRIPTION**28333 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
28334 conflict between the requirements described here and the ISO C standard is unintentional. This  
28335 volume of POSIX.1-2008 defers to the ISO C standard.28336 The `fgetwc()` function shall obtain the next character (if present) from the input stream pointed to  
28337 by `stream`, convert that to the corresponding wide-character code, and advance the associated file  
28338 position indicator for the stream (if defined).

28339 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

28340 CX The `fgetwc()` function may mark the last data access timestamp of the file associated with `stream`  
28341 for update. The last data access timestamp shall be marked for update by the first successful  
28342 execution of `fgetwc()`, `fgetws()`, `fwscanf()`, `getwc()`, `getwchar()`, `ofwscanf()`, `vfwscanf()`, or `wscanf()`  
28343 using `stream` that returns data not supplied by a prior call to `ungetwc()`.28344 **RETURN VALUE**28345 Upon successful completion, the `fgetwc()` function shall return the wide-character code of the  
28346 character read from the input stream pointed to by `stream` converted to a type `wint_t`. If the end-  
28347 of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for  
28348 the stream shall be set and `fgetwc()` shall return WEOF. If a read error occurs, the error indicator  
28349 CX for the stream shall be set, `fgetwc()` shall return WEOF, and shall set `errno` to indicate the error. If  
28350 an encoding error occurs, the error indicator for the stream shall be set, `fgetwc()` shall return  
28351 WEOF, and shall set `errno` to indicate the error.28352 **ERRORS**28353 The `fgetwc()` function shall fail if data needs to be read and:28354 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and  
28355 the thread would be delayed in the `fgetwc()` operation.28356 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for  
28357 reading.28358 **[EILSEQ]** The data obtained from the input stream does not form a valid character.28359 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data  
28360 was transferred.28361 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process  
28362 group attempting to read from its controlling terminal, and either the process  
28363 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.  
28364 This error may also be generated for implementation-defined reasons.28365 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the  
28366 offset maximum associated with the corresponding stream.

28367 The *fgetwc()* function may fail if:

28368 CX [ENOMEM] Insufficient storage space is available.

28369 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
28370 capabilities of the device.

#### 28371 EXAMPLES

28372 None.

#### 28373 APPLICATION USAGE

28374 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
28375 end-of-file condition.

#### 28376 RATIONALE

28377 None.

#### 28378 FUTURE DIRECTIONS

28379 None.

#### 28380 SEE ALSO

28381 *feof()*, *ferror()*, *fopen()*

28382 XBD <stdio.h>, <wchar.h>

#### 28383 CHANGE HISTORY

28384 First released in Issue 4. Derived from the MSE working draft.

#### 28385 Issue 5

28386 The Optional Header (OH) marking is removed from <stdio.h>.

28387 Large File Summit extensions are added.

#### 28388 Issue 6

28389 Extensions beyond the ISO C standard are marked.

28390 The following new requirements on POSIX implementations derive from alignment with the  
28391 Single UNIX Specification:

- 28392 • The [EIO] and [Eoverflow] mandatory error conditions are added.
- 28393 • The [ENOMEM] and [ENXIO] optional error conditions are added.

28394 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN]  
28395 error in the ERRORS section from “the process would be delayed” to “the thread would be  
28396 delayed”.

#### 28397 Issue 7

28398 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

28399 Changes are made related to support for finegrained timestamps.

**fgetws()**28400 **NAME**28401 `fgetws` — get a wide-character string from a stream28402 **SYNOPSIS**28403 `#include <stdio.h>`28404 `#include <wchar.h>`28405 `wchar_t *fgetws(wchar_t *restrict ws, int n,`  
28406 `FILE *restrict stream);`28407 **DESCRIPTION**28408 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
28409 conflict between the requirements described here and the ISO C standard is unintentional. This  
28410 volume of POSIX.1-2008 defers to the ISO C standard.28411 The `fgetws()` function shall read characters from the *stream*, convert these to the corresponding  
28412 wide-character codes, place them in the `wchar_t` array pointed to by *ws*, until *n*–1 characters are  
28413 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is  
28414 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character  
28415 code.

28416 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

28417 **CX** The `fgetws()` function may mark the last data access timestamp of the file associated with *stream*  
28418 for update. The last data access timestamp shall be marked for update by the first successful  
28419 execution of `fgetwc()`, `fgetws()`, `fwscanf()`, `getwc()`, `getwchar()`, `vwscanf()`, `vwsscanf()`, or `wscanf()`  
28420 using *stream* that returns data not supplied by a prior call to `ungetwc()`.28421 **RETURN VALUE**28422 Upon successful completion, `fgetws()` shall return *ws*. If the end-of-file indicator for the stream is  
28423 set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and  
28424 `fgetws()` shall return a null pointer. If a read error occurs, the error indicator for the stream shall  
28425 **CX** be set, `fgetws()` shall return a null pointer, and shall set *errno* to indicate the error.28426 **ERRORS**28427 Refer to `fgetwc()`.28428 **EXAMPLES**

28429 None.

28430 **APPLICATION USAGE**

28431 None.

28432 **RATIONALE**

28433 None.

28434 **FUTURE DIRECTIONS**

28435 None.

28436 **SEE ALSO**28437 `fopen()`, `fread()`28438 XBD `<stdio.h>`, `<wchar.h>`28439 **CHANGE HISTORY**

28440 First released in Issue 4. Derived from the MSE working draft.

- 28441 **Issue 5**  
28442 The Optional Header (OH) marking is removed from `<stdio.h>`.
- 28443 **Issue 6**  
28444 Extensions beyond the ISO C standard are marked.  
28445 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.
- 28446 **Issue 7**  
28447 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.  
28448 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fileno()**28449 **NAME**28450 `fileno` — map a stream pointer to a file descriptor28451 **SYNOPSIS**

```
28452 CX #include <stdio.h>
28453 int fileno(FILE *stream);
```

28454 **DESCRIPTION**

28455 The `fileno()` function shall return the integer file descriptor associated with the stream pointed to  
 28456 by `stream`.

28457 **RETURN VALUE**

28458 Upon successful completion, `fileno()` shall return the integer value of the file descriptor  
 28459 associated with `stream`. Otherwise, the value `-1` shall be returned and `errno` set to indicate the  
 28460 error.

28461 **ERRORS**28462 The `fileno()` function may fail if:

28463 [EBADF] The `stream` argument is not a valid stream, or the stream is not associated with  
 28464 a file.

28465 **EXAMPLES**

28466 None.

28467 **APPLICATION USAGE**

28468 None.

28469 **RATIONALE**

28470 Without some specification of which file descriptors are associated with these streams, it is  
 28471 impossible for an application to set up the streams for another application it starts with `fork()`  
 28472 and `exec`. In particular, it would not be possible to write a portable version of the `sh` command  
 28473 interpreter (although there may be other constraints that would prevent that portability).

28474 **FUTURE DIRECTIONS**

28475 None.

28476 **SEE ALSO**28477 Section 2.5.1 (on page 491), `dirfd()`, `fdopen()`, `fopen()`, `stdin`28478 XBD `<stdio.h>`28479 **CHANGE HISTORY**

28480 First released in Issue 1. Derived from Issue 1 of the SVID.

28481 **Issue 6**

28482 The following new requirements on POSIX implementations derive from alignment with the  
 28483 Single UNIX Specification:

- 28484 • The [EBADF] optional error condition is added.

28485 **Issue 7**

28486 SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

28487 **NAME**

28488 flockfile, ftrylockfile, funlockfile — stdio locking functions

28489 **SYNOPSIS**

```
28490 CX #include <stdio.h>
28491 void flockfile(FILE *file);
28492 int ftrylockfile(FILE *file);
28493 void funlockfile(FILE *file);
```

28494 **DESCRIPTION**

28495 These functions shall provide for explicit application-level locking of stdio (**FILE \***) objects.  
 28496 These functions can be used by a thread to delineate a sequence of I/O statements that are  
 28497 executed as a unit.

28498 The *flockfile()* function shall acquire for a thread ownership of a (**FILE \***) object.

28499 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE \***) object if the object is  
 28500 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

28501 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is  
 28502 undefined if a thread other than the current owner calls the *funlockfile()* function.

28503 The functions shall behave as if there is a lock count associated with each (**FILE \***) object. This  
 28504 count is implicitly initialized to zero when the (**FILE \***) object is created. The (**FILE \***) object is  
 28505 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE \***)  
 28506 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and  
 28507 the caller owns the (**FILE \***) object, the count shall be incremented. Otherwise, the calling thread  
 28508 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall  
 28509 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)  
 28510 and *funlockfile()* to be nested.

28511 All functions that reference (**FILE \***) objects shall behave as if they use *flockfile()* and *funlockfile()*  
 28512 internally to obtain ownership of these (**FILE \***) objects.

28513 **RETURN VALUE**

28514 None for *flockfile()* and *funlockfile()*.

28515 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock  
 28516 cannot be acquired.

28517 **ERRORS**

28518 No errors are defined.

28519 **EXAMPLES**

28520 None.

28521 **APPLICATION USAGE**

28522 Applications using these functions may be subject to priority inversion, as discussed in XBD  
 28523 [Section 3.285](#) (on page 79).

28524 **RATIONALE**

28525 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each  
 28526 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,  
 28527 analogous to *pthread\_mutex\_trylock()*.

28528 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.  
 28529 This both provides thread-safety of these functions without requiring a second level of internal  
 28530 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

**flockfile()**

28531 Application developers and implementors should be aware that there are potential deadlock  
 28532 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested  
 28533 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of  
 28534 implementation-defined line-buffered output streams to be flushed. If two threads each hold the  
 28535 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring  
 28536 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can  
 28537 typically be avoided by acquiring locks on input streams before locks on output streams if a  
 28538 thread would be acquiring both.

28539 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*  
 28540 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where  
 28541 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the  
 28542 `*_unlocked` functions/macros. This moves the cost/performance tradeoff to the optimal point.

28543 **FUTURE DIRECTIONS**

28544 None.

28545 **SEE ALSO**28546 [\*getc\\_unlocked\(\)\*](#)28547 XBD [Section 3.285](#) (on page 79), [`<stdio.h>`](#)28548 **CHANGE HISTORY**

28549 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

28550 **Issue 6**

28551 These functions are marked as part of the Thread-Safe Functions option.

28552 **Issue 7**28553 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions  
 28554 option to the Base.

28555 **NAME**

28556 floor, floorf, floorl — floor function

28557 **SYNOPSIS**

```
28558 #include <math.h>
28559 double floor(double x);
28560 float floorf(float x);
28561 long double floorl(long double x);
```

28562 **DESCRIPTION**

28563 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 28564 conflict between the requirements described here and the ISO C standard is unintentional. This  
 28565 volume of POSIX.1-2008 defers to the ISO C standard.

28566 These functions shall compute the largest integral value not greater than  $x$ .

28567 An application wishing to check for error situations should set *errno* to zero and call  
 28568 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 28569 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 28570 zero, an error has occurred.

28571 **RETURN VALUE**

28572 Upon successful completion, these functions shall return the largest integral value not greater  
 28573 than  $x$ , expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the  
 28574 function.

28575 MX If  $x$  is NaN, a NaN shall be returned.

28576 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

28577 XSI If the correct value would cause overflow, a range error shall occur and *floor()*, *floorf()*, and  
 28578 *floorl()* shall return the value of the macro `-HUGE_VAL`, `-HUGE_VALF`, and `-HUGE_VALL`,  
 28579 respectively.

28580 **ERRORS**

28581 These functions shall fail if:

28582 XSI **Range Error** The result would cause an overflow.

28583 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 28584 then *errno* shall be set to [ERANGE]. If the integer expression  
 28585 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 28586 floating-point exception shall be raised.

28587 **EXAMPLES**

28588 None.

28589 **APPLICATION USAGE**

28590 The integral value returned by these functions might not be expressible as an **int** or **long**. The  
 28591 return value should be tested before assigning it to an integer type to avoid the undefined  
 28592 results of an integer overflow.

28593 The *floor()* function can only overflow when the floating-point representation has  
 28594 `DBL_MANT_DIG > DBL_MAX_EXP`.

28595 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 28596 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**floor()**28597 **RATIONALE**

28598 None.

28599 **FUTURE DIRECTIONS**

28600 None.

28601 **SEE ALSO**28602 *ceil()*, *feclearexcept()*, *fetestexcept()*, *isnan()*28603 [Section 4.19](#) (on page 116), `<math.h>`28604 **CHANGE HISTORY**

28605 First released in Issue 1. Derived from Issue 1 of the SVID.

28606 **Issue 5**28607 The DESCRIPTION is updated to indicate how an application should check for an error. This  
28608 text was previously published in the APPLICATION USAGE section.28609 **Issue 6**28610 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.28611 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
28612 revised to align with the ISO/IEC 9899:1999 standard.28613 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
28614 marked.

28615 **NAME**

28616 fma, fmaf, fmal — floating-point multiply-add

28617 **SYNOPSIS**

28618 #include &lt;math.h&gt;

28619 double fma(double x, double y, double z);

28620 float fmaf(float x, float y, float z);

28621 long double fmal(long double x, long double y, long double z);

28622 **DESCRIPTION**28623 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
28624 conflict between the requirements described here and the ISO C standard is unintentional. This  
28625 volume of POSIX.1-2008 defers to the ISO C standard.28626 These functions shall compute  $(x * y) + z$ , rounded as one ternary operation; they shall compute  
28627 the value (as if) to infinite precision and round once to the result format, according to the  
28628 rounding mode characterized by the value of FLT\_ROUNDS.28629 An application wishing to check for error situations should set *errno* to zero and call  
28630 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
28631 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
28632 zero, an error has occurred.28633 **RETURN VALUE**28634 Upon successful completion, these functions shall return  $(x * y) + z$ , rounded as one ternary  
28635 operation.28636 **MX** If the result overflows or underflows, a range error may occur. On systems that support the IEC  
28637 60559 Floating-Point option, if the result overflows a range error shall occur.28638 If *x* or *y* are NaN, a NaN shall be returned.28639 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a  
28640 domain error shall occur, and either a NaN (if supported), or an implementation-defined value  
28641 shall be returned.28642 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and  
28643 either a NaN (if supported), or an implementation-defined value shall be returned.28644 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a  
28645 domain error may occur.28646 If  $x*y$  is not  $0*Inf$  nor  $Inf*0$  and *z* is a NaN, a NaN shall be returned.28647 **ERRORS**

28648 These functions shall fail if:

28649 **MX** **Domain Error** The value of  $x*y+z$  is invalid, or the value  $x*y$  is invalid and *z* is not a NaN.28650 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
28651 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
28652 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
28653 shall be raised.28654 **MX** **Range Error** The result overflows.28655 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
28656 then *errno* shall be set to [ERANGE]. If the integer expression  
28657 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
28658 floating-point exception shall be raised.

**fma()**

28659 These functions may fail if:

|       |    |                     |                                                                                                                                                                                                                                                                                                                            |
|-------|----|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 28660 | MX | <b>Domain Error</b> | The value $x*y$ is invalid and $z$ is a NaN.<br><br>If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised. |
| 28665 |    | <b>Range Error</b>  | The result underflows.<br><br>If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [ERANGE]. If the integer expression ( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.                   |
| 28670 |    | <b>Range Error</b>  | The result overflows.<br><br>If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [ERANGE]. If the integer expression ( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.                     |

28675 **EXAMPLES**

28676 None.

28677 **APPLICATION USAGE**

28678 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
28679 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28680 **RATIONALE**

28681 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its  
28682 unexpected use by the compiler can undermine carefully written code. The FP\_CONTRACT  
28683 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees  
28684 its use where desired. Many current machines provide hardware floating multiply-add  
28685 instructions; software implementation can be used for others.

28686 **FUTURE DIRECTIONS**

28687 None.

28688 **SEE ALSO**

28689 [feclearexcept\(\)](#), [fecetestexcept\(\)](#)

28690 XBD [Section 4.19](#) (on page 116), [<math.h>](#)

28691 **CHANGE HISTORY**

28692 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28693 **Issue 7**

28694 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied,  
28695 adding a “may fail” range error for non-MX systems.

28696 **NAME**

28697 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

28698 **SYNOPSIS**

28699 #include &lt;math.h&gt;

28700 double fmax(double x, double y);

28701 float fmaxf(float x, float y);

28702 long double fmaxl(long double x, long double y);

28703 **DESCRIPTION**28704 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
28705 conflict between the requirements described here and the ISO C standard is unintentional. This  
28706 volume of POSIX.1-2008 defers to the ISO C standard.28707 MX These functions shall determine the maximum numeric value of their arguments. NaN  
28708 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
28709 then these functions shall choose the numeric value.28710 **RETURN VALUE**28711 Upon successful completion, these functions shall return the maximum numeric value of their  
28712 arguments.

28713 MX If just one argument is a NaN, the other argument shall be returned.

28714 If *x* and *y* are NaN, a NaN shall be returned.28715 **ERRORS**

28716 No errors are defined.

28717 **EXAMPLES**

28718 None.

28719 **APPLICATION USAGE**

28720 None.

28721 **RATIONALE**

28722 None.

28723 **FUTURE DIRECTIONS**

28724 None.

28725 **SEE ALSO**28726 *fdim()*, *fmin()*

28727 XBD &lt;math.h&gt;

28728 **CHANGE HISTORY**

28729 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28730 **Issue 7**

28731 Austin Group Interpretation 1003.1-2001 #007 is applied.

**fmemopen()**28732 **NAME**28733 `fmemopen` — open a memory buffer stream28734 **SYNOPSIS**

```
28735 CX #include <stdio.h>
28736 FILE *fmemopen(void *restrict buf, size_t size,
28737               const char *restrict mode);
```

28738 **DESCRIPTION**

28739 The `fmemopen()` function shall associate the buffer given by the `buf` and `size` arguments with a  
 28740 stream. The `buf` argument shall be either a null pointer or point to a buffer that is at least `size`  
 28741 bytes long.

28742 The `mode` argument is a character string having one of the following values:

|       |                                                         |                                                                                                                    |
|-------|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| 28743 | <code>r</code> or <code>rb</code>                       | Open the stream for reading.                                                                                       |
| 28744 | <code>w</code> or <code>wb</code>                       | Open the stream for writing.                                                                                       |
| 28745 | <code>a</code> or <code>ab</code>                       | Append; open the stream for writing at the first null byte.                                                        |
| 28746 | <code>r+</code> or <code>rb+</code> or <code>r+b</code> | Open the stream for update (reading and writing).                                                                  |
| 28747 | <code>w+</code> or <code>wb+</code> or <code>w+b</code> | Open the stream for update (reading and writing). Truncate the buffer<br>28748 contents.                           |
| 28749 | <code>a+</code> or <code>ab+</code> or <code>a+b</code> | Append; open the stream for update (reading and writing); the initial<br>28750 position is at the first null byte. |

28751 The character 'b' shall have no effect.

28752 If a null pointer is specified as the `buf` argument, `fmemopen()` shall allocate `size` bytes of memory  
 28753 as if by a call to `malloc()`. This buffer shall be automatically freed when the stream is closed.  
 28754 Because this feature is only useful when the stream is opened for updating (because there is no  
 28755 way to get a pointer to the buffer) the `fmemopen()` call may fail if the `mode` argument does not  
 28756 include a '+'. .

28757 The stream maintains a current position in the buffer. This position is initially set to either the  
 28758 beginning of the buffer (for `r` and `w` modes) or to the first null byte in the buffer (for `a` modes). If  
 28759 no null byte is found in append mode, the initial position is set to one byte after the end of the  
 28760 buffer.

28761 If `buf` is a null pointer, the initial position shall always be set to the beginning of the buffer.

28762 The stream also maintains the size of the current buffer contents. For modes `r` and `r+` the size is  
 28763 set to the value given by the `size` argument. For modes `w` and `w+` the initial size is zero and for  
 28764 modes `a` and `a+` the initial size is either the position of the first null byte in the buffer or the value  
 28765 of the `size` argument if no null byte is found.

28766 A read operation on the stream cannot advance the current buffer position beyond the current  
 28767 buffer size. Reaching the buffer size in a read operation counts as "end-of-file". Null bytes in the  
 28768 buffer have no special meaning for reads. The read operation starts at the current buffer position  
 28769 of the stream.

28770 A write operation starts either at the current position of the stream (if mode has not specified  
 28771 'a' as the first character) or at the current size of the stream (if mode had 'a' as the first  
 28772 character). If the current position at the end of the write is larger than the current buffer size, the  
 28773 current buffer size is set to the current position. A write operation on the stream cannot advance  
 28774 the current buffer size beyond the size given in the `size` argument.

28775 When a stream open for writing is flushed or closed, a null byte is written at the current position  
 28776 or at the end of the buffer, depending on the size of the contents. If a stream open for update is  
 28777 flushed or closed and the last write has advanced the current buffer size, a null byte is written at  
 28778 the end of the buffer if it fits.

28779 An attempt to seek a memory buffer stream to a negative position or to a position larger than the  
 28780 buffer size given in the *size* argument shall fail.

#### 28781 RETURN VALUE

28782 Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the  
 28783 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

#### 28784 ERRORS

28785 The *fmemopen()* function shall fail if:

28786 [EINVAL] The *size* argument specifies a buffer size of zero.

28787 The *fmemopen()* function may fail if:

28788 [EINVAL] The value of the *mode* argument is not valid.

28789 [EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a  
 28790 '+' character.

28791 [ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *size*  
 28792 has failed.

28793 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

#### 28794 EXAMPLES

```
28795 #include <stdio.h>
28796 #include <string.h>
28797 static char buffer[] = "foobar";
28798 int
28799 main (void)
28800 {
28801     int ch;
28802     FILE *stream;
28803     stream = fmemopen(buffer, strlen (buffer), "r");
28804     if (stream == NULL)
28805         /* handle error */;
28806     while ((ch = fgetc(stream)) != EOF)
28807         printf("Got %c\n", ch);
28808     fclose(stream);
28809     return (0);
28810 }
```

28811 This program produces the following output:

```
28812 Got f
28813 Got o
28814 Got o
28815 Got b
28816 Got a
28817 Got r
```

**fmemopen()***System Interfaces*28818 **APPLICATION USAGE**

28819 None.

28820 **RATIONALE**28821 This interface has been introduced to eliminate many of the errors encountered in the  
28822 construction of strings, notably overflowing of strings. This interface prevents overflow.28823 **FUTURE DIRECTIONS**

28824 None.

28825 **SEE ALSO**28826 *fdopen(), fopen(), freopen(), malloc(), open\_memstream()*28827 XBD `<stdio.h>`28828 **CHANGE HISTORY**

28829 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

28830 **NAME**

28831 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

28832 **SYNOPSIS**

28833 #include &lt;math.h&gt;

28834 double fmin(double x, double y);

28835 float fminf(float x, float y);

28836 long double fminl(long double x, long double y);

28837 **DESCRIPTION**28838 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
28839 conflict between the requirements described here and the ISO C standard is unintentional. This  
28840 volume of POSIX.1-2008 defers to the ISO C standard.28841 MX These functions shall determine the minimum numeric value of their arguments. NaN  
28842 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
28843 then these functions shall choose the numeric value.28844 **RETURN VALUE**28845 Upon successful completion, these functions shall return the minimum numeric value of their  
28846 arguments.

28847 MX If just one argument is a NaN, the other argument shall be returned.

28848 If *x* and *y* are NaN, a NaN shall be returned.28849 **ERRORS**

28850 No errors are defined.

28851 **EXAMPLES**

28852 None.

28853 **APPLICATION USAGE**

28854 None.

28855 **RATIONALE**

28856 None.

28857 **FUTURE DIRECTIONS**

28858 None.

28859 **SEE ALSO**28860 *fdim()*, *fmax()*

28861 XBD &lt;math.h&gt;

28862 **CHANGE HISTORY**

28863 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28864 **Issue 7**

28865 Austin Group Interpretation 1003.1-2001 #008 is applied.

**fmod()**28866 **NAME**28867 `fmod, fmodf, fmodl` — floating-point remainder value function28868 **SYNOPSIS**28869 `#include <math.h>`28870 `double fmod(double x, double y);`28871 `float fmodf(float x, float y);`28872 `long double fmodl(long double x, long double y);`28873 **DESCRIPTION**28874 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
28875 conflict between the requirements described here and the ISO C standard is unintentional. This  
28876 volume of POSIX.1-2008 defers to the ISO C standard.28877 These functions shall return the floating-point remainder of the division of  $x$  by  $y$ .28878 An application wishing to check for error situations should set *errno* to zero and call  
28879 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
28880 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
28881 zero, an error has occurred.28882 **RETURN VALUE**28883 These functions shall return the value  $x-i*y$ , for some integer  $i$  such that, if  $y$  is non-zero, the  
28884 result has the same sign as  $x$  and magnitude less than the magnitude of  $y$ .28885 If the correct value would cause underflow, and is not representable, a range error may occur,  
28886 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.28887 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.28888 If  $y$  is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-  
28889 defined value shall be returned.28890 If  $x$  is infinite, a domain error shall occur, and either a NaN (if supported), or an  
28891 implementation-defined value shall be returned.28892 If  $x$  is  $\pm 0$  and  $y$  is not zero,  $\pm 0$  shall be returned.28893 If  $x$  is not infinite and  $y$  is  $\pm \text{Inf}$ ,  $x$  shall be returned.28894 If the correct value would cause underflow, and is representable, a range error may occur and  
28895 the correct value shall be returned.28896 **ERRORS**

28897 These functions shall fail if:

28898 **MX** **Domain Error** The  $x$  argument is infinite or  $y$  is zero.28899 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
28900 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
28901 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
28902 shall be raised.

28903 These functions may fail if:

28904 **Range Error** The result underflows.28905 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
28906 then *errno* shall be set to [ERANGE]. If the integer expression  
28907 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
28908 floating-point exception shall be raised.

28909 **EXAMPLES**

28910 None.

28911 **APPLICATION USAGE**28912 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
28913 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.28914 **RATIONALE**

28915 None.

28916 **FUTURE DIRECTIONS**

28917 None.

28918 **SEE ALSO**28919 *feclearexcept()*, *fetestexcept()*, *isnan()*

28920 Section 4.19 (on page 116), &lt;math.h&gt;

28921 **CHANGE HISTORY**

28922 First released in Issue 1. Derived from Issue 1 of the SVID.

28923 **Issue 5**28924 The DESCRIPTION is updated to indicate how an application should check for an error. This  
28925 text was previously published in the APPLICATION USAGE section.28926 **Issue 6**28927 The behavior for when the *y* argument is zero is now defined.28928 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999  
28929 standard.28930 The DESCRIPTION, RETURN VALUE ERRORS, and APPLICATION USAGE sections are  
28931 revised to align with the ISO/IEC 9899:1999 standard.28932 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
28933 marked.

**fmtmsg()**28934 **NAME**28935 `fmtmsg` — display a message in the specified format on standard error and/or a system console28936 **SYNOPSIS**

```
28937 XSI #include <fmtmsg.h>
28938
28938 int fmtmsg(long classification, const char *label, int severity,
28939            const char *text, const char *action, const char *tag);
```

28940 **DESCRIPTION**28941 The `fmtmsg()` function shall display messages in a specified format instead of the traditional  
28942 `printf()` function.28943 Based on a message's classification component, `fmtmsg()` shall write a formatted message either  
28944 to standard error, to the console, or to both.28945 A formatted message consists of up to five components as defined below. The component  
28946 *classification* is not part of a message displayed to the user, but defines the source of the message  
28947 and directs the display of the formatted message.28948 *classification* Contains the sum of identifying values constructed from the constants defined  
28949 below. Any one identifier from a subclass may be used in combination with a  
28950 single identifier from a different subclass. Two or more identifiers from the  
28951 same subclass should not be used together, with the exception of identifiers  
28952 from the display subclass. (Both display subclass identifiers may be used so  
28953 that messages can be displayed to both standard error and the system  
28954 console.)28955 **Major Classifications**28956 Identifies the source of the condition. Identifiers are: MM\_HARD  
28957 (hardware), MM\_SOFT (software), and MM\_FIRM (firmware).28958 **Message Source Subclassifications**28959 Identifies the type of software in which the problem is detected.  
28960 Identifiers are: MM\_APPL (application), MM\_UTIL (utility), and  
28961 MM\_OPSYS (operating system).28962 **Display Subclassifications**28963 Indicates where the message is to be displayed. Identifiers are:  
28964 MM\_PRINT to display the message on the standard error stream,  
28965 MM\_CONSOLE to display the message on the system console. One or  
28966 both identifiers may be used.28967 **Status Subclassifications**28968 Indicates whether the application can recover from the condition.  
28969 Identifiers are: MM\_RECOVER (recoverable) and MM\_NRECOV (non-  
28970 recoverable).28971 An additional identifier, MM\_NULLMC, indicates that no classification  
28972 component is supplied for the message.28973 *label* Identifies the source of the message. The format is two fields separated by a  
28974 <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.28975 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*  
28976 are:

|       |               |            |                                                                                                                                                                                                                     |
|-------|---------------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 28977 |               | MM_HALT    | Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".                                                                                                           |
| 28978 |               |            |                                                                                                                                                                                                                     |
| 28979 |               | MM_ERROR   | Indicates that the application has detected a fault. Produces the string "ERROR".                                                                                                                                   |
| 28980 |               |            |                                                                                                                                                                                                                     |
| 28981 |               | MM_WARNING | Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".                                                                                   |
| 28982 |               |            |                                                                                                                                                                                                                     |
| 28983 |               |            |                                                                                                                                                                                                                     |
| 28984 |               | MM_INFO    | Provides information about a condition that is not in error. Produces the string "INFO".                                                                                                                            |
| 28985 |               |            |                                                                                                                                                                                                                     |
| 28986 |               | MM_NOSEV   | Indicates that no severity level is supplied for the message.                                                                                                                                                       |
| 28987 | <i>text</i>   |            | Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.                           |
| 28988 |               |            |                                                                                                                                                                                                                     |
| 28989 |               |            |                                                                                                                                                                                                                     |
| 28990 | <i>action</i> |            | Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size. |
| 28991 |               |            |                                                                                                                                                                                                                     |
| 28992 |               |            |                                                                                                                                                                                                                     |
| 28993 | <i>tag</i>    |            | An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".            |
| 28994 |               |            |                                                                                                                                                                                                                     |
| 28995 |               |            |                                                                                                                                                                                                                     |

28996 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg()*  
 28997 which message components it is to select when writing messages to standard error. The value of  
 28998 *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*,  
 28999 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the  
 29000 component's value is not the component's null value, *fmtmsg()* shall include that component in  
 29001 the message when writing the message to standard error. If *MSGVERB* does not include a  
 29002 keyword for a message component, that component shall not be included in the display of the  
 29003 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the  
 29004 null string, if its value is not of the correct format, or if it contains keywords other than the valid  
 29005 ones listed above, *fmtmsg()* shall select all components.

29006 *MSGVERB* shall determine which components are selected for display to standard error. All  
 29007 message components shall be included in console messages.

#### 29008 RETURN VALUE

29009 The *fmtmsg()* function shall return one of the following values:

|       |          |                                                                                           |
|-------|----------|-------------------------------------------------------------------------------------------|
| 29010 | MM_OK    | The function succeeded.                                                                   |
| 29011 | MM_NOTOK | The function failed completely.                                                           |
| 29012 | MM_NOMSG | The function was unable to generate a message on standard error, but otherwise succeeded. |
| 29013 |          |                                                                                           |
| 29014 | MM_NOCON | The function was unable to generate a console message, but otherwise succeeded.           |
| 29015 |          |                                                                                           |

#### 29016 ERRORS

29017 None.

**fmtmsg()**29018 **EXAMPLES**

29019 1. The following example of *fmtmsg()*:

```
29020 fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",
29021 "refer to cat in user's reference manual", "XSI:cat:001")
```

29022 produces a complete message in the specified message format:

```
29023 XSI:cat: ERROR: illegal option
29024 TO FIX: refer to cat in user's reference manual XSI:cat:001
```

29025 2. When the environment variable *MSGVERB* is set as follows:

```
29026 MSGVERB=severity:text:action
```

29027 and Example 1 is used, *fmtmsg()* produces:

```
29028 ERROR: illegal option
29029 TO FIX: refer to cat in user's reference manual
```

29030 **APPLICATION USAGE**

29031 One or more message components may be systematically omitted from messages generated by  
29032 an application by using the null value of the argument for that component.

29033 **RATIONALE**

29034 None.

29035 **FUTURE DIRECTIONS**

29036 None.

29037 **SEE ALSO**

29038 [\*fprintf\(\)\*](#)

29039 XBD [<fmtmsg.h>](#)

29040 **CHANGE HISTORY**

29041 First released in Issue 4, Version 2.

29042 **Issue 5**

29043 Moved from X/OPEN UNIX extension to BASE.

29044 **NAME**

29045 fnmatch — match a filename or a pathname

29046 **SYNOPSIS**

29047 #include &lt;fnmatch.h&gt;

29048 int fnmatch(const char \*pattern, const char \*string, int flags);

29049 **DESCRIPTION**

29050 The *fnmatch()* function shall match patterns as described in XCU Section 2.13.1 (on page 2332)  
 29051 and Section 2.13.2 (on page 2332). It checks the string specified by the *string* argument to see if it  
 29052 matches the pattern specified by the *pattern* argument.

29053 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-  
 29054 inclusive OR of zero or more of the flags defined in <fnmatch.h>. If the FNM\_PATHNAME flag  
 29055 is set in *flags*, then a <slash> character ('/') in *string* shall be explicitly matched by a <slash> in  
 29056 *pattern*; it shall not be matched by either the <asterisk> or <question-mark> special characters,  
 29057 nor by a bracket expression. If the FNM\_PATHNAME flag is not set, the <slash> character shall  
 29058 be treated as an ordinary character.

29059 If FNM\_NOESCAPE is not set in *flags*, a <backslash> character in *pattern* followed by any other  
 29060 character shall match that second character in *string*. In particular, "\\\" shall match a  
 29061 <backslash> in *string*. If FNM\_NOESCAPE is set, a <backslash> character shall be treated as an  
 29062 ordinary character.

29063 If FNM\_PERIOD is set in *flags*, then a leading <period> ('.') in *string* shall match a <period> in  
 29064 *pattern*; as described by rule 2 in XCU Section 2.13.3 (on page 2333) where the location of  
 29065 "leading" is indicated by the value of FNM\_PATHNAME:

- 29066 • If FNM\_PATHNAME is set, a <period> is "leading" if it is the first character in *string* or if  
 29067 it immediately follows a <slash>.
- 29068 • If FNM\_PATHNAME is not set, a <period> is "leading" only if it is the first character of  
 29069 *string*.

29070 If FNM\_PERIOD is not set, then no special restrictions are placed on matching a period.

29071 **RETURN VALUE**

29072 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no  
 29073 match, *fnmatch()* shall return FNM\_NOMATCH, which is defined in <fnmatch.h>. If an error  
 29074 occurs, *fnmatch()* shall return another non-zero value.

29075 **ERRORS**

29076 No errors are defined.

29077 **EXAMPLES**

29078 None.

29079 **APPLICATION USAGE**

29080 The *fnmatch()* function has two major uses. It could be used by an application or utility that  
 29081 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of  
 29082 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that  
 29083 need to match strings in a similar manner.

29084 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The  
 29085 default action of this function is to match filenames, rather than pathnames, since it gives no  
 29086 special significance to the <slash> character. With the FNM\_PATHNAME flag, *fnmatch()* does  
 29087 match pathnames, but without tilde expansion, parameter expansion, or special treatment for a  
 29088 <period> at the beginning of a filename.

**fnmatch()**29089 **RATIONALE**

29090 This function replaced the REG\_FILENAME flag of *regcomp()* in early proposals of this volume  
29091 of POSIX.1-2008. It provides virtually the same functionality as the *regcomp()* and *regexec()*  
29092 functions using the REG\_FILENAME and REG\_FSLASH flags (the REG\_FSLASH flag was  
29093 proposed for *regcomp()*, and would have had the opposite effect from FNM\_PATHNAME), but  
29094 with a simpler function and less system overhead.

29095 **FUTURE DIRECTIONS**

29096 None.

29097 **SEE ALSO**

29098 [glob\(\)](#), Section 2.6

29099 XBD [<fnmatch.h>](#)

29100 **CHANGE HISTORY**

29101 First released in Issue 4. Derived from the ISO POSIX-2 standard.

29102 **Issue 5**

29103 Moved from POSIX2 C-language Binding to BASE.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

29104 **NAME**

29105 fopen — open a stream

29106 **SYNOPSIS**

29107 #include &lt;stdio.h&gt;

29108 FILE \*fopen(const char \*restrict filename, const char \*restrict mode);

29109 **DESCRIPTION**

29110 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29111 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29112 volume of POSIX.1-2008 defers to the ISO C standard.

29113 The *fopen()* function shall open the file whose pathname is the string pointed to by *filename*, and  
 29114 associates a stream with it.

29115 The *mode* argument points to a string. If the string is one of the following, the file shall be opened  
 29116 in the indicated mode. Otherwise, the behavior is undefined.

29117 *r* or *rb* Open file for reading.

29118 *w* or *wb* Truncate to zero length or create file for writing.

29119 *a* or *ab* Append; open or create file for writing at end-of-file.

29120 *r+* or *rb+* or *r+b* Open file for update (reading and writing).

29121 *w+* or *wb+* or *w+b* Truncate to zero length or create file for update.

29122 *a+* or *ab+* or *a+b* Append; open or create file for update, writing at end-of-file.

29123 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening  
 29124 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not  
 29125 exist or cannot be read.

29126 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all  
 29127 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening  
 29128 calls to *fseek()*.

29129 When a file is opened with update mode ('+' as the second or third character in the *mode*  
 29130 argument), both input and output may be performed on the associated stream. However, the  
 29131 application shall ensure that output is not directly followed by input without an intervening call  
 29132 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly  
 29133 followed by output without an intervening call to a file positioning function, unless the input  
 29134 operation encounters end-of-file.

29135 When opened, a stream is fully buffered if and only if it can be determined not to refer to an  
 29136 interactive device. The error and end-of-file indicators for the stream shall be cleared.

29137 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon  
 29138 successful completion, *fopen()* shall mark for update the last data access, last data modification,  
 29139 and last file status change timestamps of the file and the last file status change and last data  
 29140 modification timestamps of the parent directory.

29141 If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, the  
 29142 *fopen()* function shall create a file as if it called the *creat()* function with a value appropriate for  
 29143 the *path* argument interpreted from *filename* and a value of S\_IRUSR | S\_IWUSR | S\_IRGRP |  
 29144 S\_IWGRP | S\_IROTH | S\_IWOTH for the *mode* argument.

29145 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,  
 29146 *fopen()* shall mark for update the last data modification and last file status change timestamps of

# fopen()

29147 the file.

29148 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the

29149 encoding rule shall be cleared, and the associated *mbstate\_t* object shall be set to describe an

29150 initial conversion state.

29151 CX The file descriptor associated with the opened stream shall be allocated and opened as if by a

29152 call to *open()* with the following flags:

| <i>fopen()</i> Mode                   | <i>open()</i> Flags       |
|---------------------------------------|---------------------------|
| <i>r</i> or <i>rb</i>                 | O_RDONLY                  |
| <i>w</i> or <i>wb</i>                 | O_WRONLY O_CREAT O_TRUNC  |
| <i>a</i> or <i>ab</i>                 | O_WRONLY O_CREAT O_APPEND |
| <i>r+</i> or <i>rb+</i> or <i>r+b</i> | O_RDWR                    |
| <i>w+</i> or <i>wb+</i> or <i>w+b</i> | O_RDWR O_CREAT O_TRUNC    |
| <i>a+</i> or <i>ab+</i> or <i>a+b</i> | O_RDWR O_CREAT O_APPEND   |

## RETURN VALUE

29160 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.

29161

29162 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

## ERRORS

29163 The *fopen()* function shall fail if:

29165 CX [EACCES] Search permission is denied on a component of the path prefix, or the file

29166 exists and the permissions specified by *mode* are denied, or the file does not

29167 exist and write permission is denied for the parent directory of the file to be

29168 created.

29169 CX [EINTR] A signal was caught during *fopen()*.

29170 CX [EISDIR] The named file is a directory and *mode* requires write access.

29171 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*

29172 argument.

29173 CX [EMFILE] All file descriptors available to the process are currently open.

29174 CX [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

29175 CX [ENAMETOOLONG]

29176 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a

29177 symbolic link produced an intermediate result with a length that exceeds

29178 {PATH\_MAX}.

29179 CX [ENFILE] The maximum allowable number of files is currently open in the system.

29180 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty

29181 string.

29182 CX [ENOSPC] The directory or file system that would contain the new file cannot be

29183 expanded, the file does not exist, and the file was to be created.

29184 CX [ENOTDIR] A component of the path prefix is not a directory, or the *filename* argument

29185 contains at least one non-*<slash>* character and ends with one or more trailing

29186 *<slash>* characters and the last pathname component names an existing file

29187 that is neither a directory nor a symbolic link to a directory.

- 29188 CX [ENXIO] The named file is a character special or block special file, and the device  
29189 associated with this special file does not exist.
- 29190 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented  
29191 correctly in an object of type `off_t`.
- 29192 CX [EROFS] The named file resides on a read-only file system and *mode* requires write  
29193 access.
- 29194 The *fopen()* function may fail if:
- 29195 CX [EINVAL] The value of the *mode* argument is not valid.
- 29196 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
29197 resolution of the *path* argument.
- 29198 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.
- 29199 CX [ENAMETOOLONG]  
29200 The length of a component of a pathname is longer than {NAME\_MAX}.
- 29201 CX [ENOMEM] Insufficient storage space is available.
- 29202 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode*  
29203 requires write access.

## 29204 EXAMPLES

### 29205 Opening a File

29206 The following example tries to open the file named **file** for reading. The *fopen()* function returns  
29207 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the  
29208 file, it just ignores it.

```
29209 #include <stdio.h>
29210 ...
29211 FILE *fp;
29212 ...
29213 void rgrep(const char *file)
29214 {
29215     ...
29216     if ((fp = fopen(file, "r")) == NULL)
29217         return;
29218     ...
29219 }
```

### 29220 APPLICATION USAGE

29221 None.

### 29222 RATIONALE

29223 None.

### 29224 FUTURE DIRECTIONS

29225 None.

### 29226 SEE ALSO

29227 [creat\(\)](#), [fclose\(\)](#), [fdopen\(\)](#), [fmemopen\(\)](#), [freopen\(\)](#), [open\\_memstream\(\)](#)

29228 XBD [<stdio.h>](#)

**fopen()**29229 **CHANGE HISTORY**

29230 First released in Issue 1. Derived from Issue 1 of the SVID.

29231 **Issue 5**

29232 Large File Summit extensions are added.

29233 **Issue 6**

29234 Extensions beyond the ISO C standard are marked.

29235 The following new requirements on POSIX implementations derive from alignment with the  
29236 Single UNIX Specification:

- 29237 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
29238 file description. This change is to support large files.
- 29239 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
29240 large files.
- 29241 • The [ELOOP] mandatory error condition is added.
- 29242 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional  
29243 error conditions are added.

29244 The normative text is updated to avoid use of the term “must” for application requirements.

29245 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 29246 • The prototype for *fopen()* is updated.
- 29247 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other  
29248 than those listed, then the behavior is undefined.

29249 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
29250 [ELOOP] error condition is added.

29251 **Issue 7**

29252 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

29253 Austin Group Interpretation 1003.1-2001 #143 is applied.

29254 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set  
29255 on the open file description.

29256 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

29257 SD5-XSH-ERN-149 is applied, changing the {STREAM\_MAX} [EMFILE] error condition from a  
29258 “may fail” to a “shall fail”.

29259 Changes are made related to support for finegrained timestamps.

29260 **NAME**

29261 fork — create a new process

29262 **SYNOPSIS**

29263 #include &lt;unistd.h&gt;

29264 pid\_t fork(void);

29265 **DESCRIPTION**29266 The *fork()* function shall create a new process. The new process (child process) shall be an exact  
29267 copy of the calling process (parent process) except as detailed below:

- 29268 • The child process shall have a unique process ID.
- 29269 • The child process ID also shall not match any active process group ID.
- 29270 • The child process shall have a different parent process ID, which shall be the process ID of  
29271 the calling process.
- 29272 • The child process shall have its own copy of the parent's file descriptors. Each of the  
29273 child's file descriptors shall refer to the same open file description with the corresponding  
29274 file descriptor of the parent.
- 29275 • The child process shall have its own copy of the parent's open directory streams. Each  
29276 open directory stream in the child process may share directory stream positioning with the  
29277 corresponding directory stream of the parent.
- 29278 • The child process shall have its own copy of the parent's message catalog descriptors.
- 29279 • The child process values of *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime* shall be set to  
29280 0.
- 29281 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be  
29282 canceled; see *alarm()*.
- 29283 XSI • All *semadj* values shall be cleared.
- 29284 • File locks set by the parent process shall not be inherited by the child process.
- 29285 • The set of signals pending for the child process shall be initialized to the empty set.
- 29286 XSI • Interval timers shall be reset in the child process.
- 29287 • Any semaphores that are open in the parent process shall also be open in the child process.
- 29288 ML • The child process shall not inherit any address space memory locks established by the  
29289 parent process via calls to *mlockall()* or *mlock()*.
- 29290 • Memory mappings created in the parent shall be retained in the child process.  
29291 MAP\_PRIVATE mappings inherited from the parent shall also be MAP\_PRIVATE  
29292 mappings in the child, and any modifications to the data in these mappings made by the  
29293 parent prior to calling *fork()* shall be visible to the child. Any modifications to the data in  
29294 MAP\_PRIVATE mappings made by the parent after *fork()* returns shall be visible only to  
29295 the parent. Modifications to the data in MAP\_PRIVATE mappings made by the child shall  
29296 be visible only to the child.
- 29297 PS • For the SCHED\_FIFO and SCHED\_RR scheduling policies, the child process shall inherit  
29298 the policy and priority settings of the parent process during a *fork()* function. For other  
29299 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.

**fork()**

- 29300
- Per-process timers created by the parent shall not be inherited by the child process.
- 29301 MSG
- The child process shall have its own copy of the message queue descriptors of the parent. Each of the message descriptors of the child shall refer to the same open message queue description as the corresponding message descriptor of the parent.
- 29302
- 29303
- 29304
- No asynchronous input or asynchronous output operations shall be inherited by the child process. Any use of asynchronous control blocks created by the parent produces undefined behavior.
- 29305
- 29306
- 29307
- A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the *exec* functions is called. Fork handlers may be established by means of the *pthread\_atfork()* function in order to maintain application invariants across *fork()* calls.
- 29308
- 29309
- 29310
- 29311
- 29312
- 29313
- 29314
- 29315
- When the application calls *fork()* from a signal handler and any of the fork handlers registered by *pthread\_atfork()* calls a function that is not async-signal-safe, the behavior is undefined.
- 29316 OB TRC TRI
- If the Trace option and the Trace Inherit option are both supported:
- 29317
- 29318
- 29319
- 29320
- 29321
- 29322
- 29323
- If the calling process was being traced in a trace stream that had its inheritance policy set to *POSIX\_TRACE\_INHERITED*, the child process shall be traced into that trace stream, and the child process shall inherit the parent's mapping of trace event names to trace event type identifiers. If the trace stream in which the calling process was being traced had its inheritance policy set to *POSIX\_TRACE\_CLOSE\_FOR\_CHILD*, the child process shall not be traced into that trace stream. The inheritance policy is set by a call to the *posix\_trace\_attr\_setinherited()* function.
- 29324 OB TRC
- If the Trace option is supported, but the Trace Inherit option is not supported:
- 29325
- The child process shall not be traced into any of the trace streams of its parent process.
- 29326 OB TRC
- If the Trace option is supported, the child process of a trace controller process shall not control the trace streams controlled by its parent process.
- 29327
- 29328 CPT
- The initial value of the CPU-time clock of the child process shall be set to zero.
- 29329 TCT
- The initial value of the CPU-time clock of the single thread of the child process shall be set to zero.
- 29330
- 29331
- 29332
- 29333
- All other process characteristics defined by POSIX.1-2008 shall be the same in the parent and child processes. The inheritance of process characteristics not defined by POSIX.1-2008 is unspecified by POSIX.1-2008.
- 29334
- 29335
- After *fork()*, both the parent and the child processes shall be capable of executing independently before either one terminates.
- 29336 **RETURN VALUE**
- 29337
- 29338
- 29339
- 29340
- Upon successful completion, *fork()* shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the *fork()* function. Otherwise, -1 shall be returned to the parent process, no child process shall be created, and *errno* shall be set to indicate the error.

29341 **ERRORS**29342 The *fork()* function shall fail if:

29343 [EAGAIN] The system lacked the necessary resources to create another process, or the  
 29344 system-imposed limit on the total number of processes under execution  
 29345 system-wide or by a single user {CHILD\_MAX} would be exceeded.

29346 The *fork()* function may fail if:

29347 [ENOMEM] Insufficient storage space is available.

29348 **EXAMPLES**

29349 None.

29350 **APPLICATION USAGE**

29351 None.

29352 **RATIONALE**

29353 Many historical implementations have timing windows where a signal sent to a process group  
 29354 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the  
 29355 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of  
 29356 pending signals. This volume of POSIX.1-2008 does not require, or even permit, this behavior.  
 29357 However, it is pragmatic to expect that problems of this nature may continue to exist in  
 29358 implementations that appear to conform to this volume of POSIX.1-2008 and pass available  
 29359 verification suites. This behavior is only a consequence of the implementation failing to make  
 29360 the interval between signal generation and delivery totally invisible. From the application's  
 29361 perspective, a *fork()* call should appear atomic. A signal that is generated prior to the *fork()*  
 29362 should be delivered prior to the *fork()*. A signal sent to the process group after the *fork()* should  
 29363 be delivered to both parent and child. The implementation may actually initialize internal data  
 29364 structures corresponding to the child's set of pending signals to include signals sent to the  
 29365 process group during the *fork()*. Since the *fork()* call can be considered as atomic from the  
 29366 application's perspective, the set would be initialized as empty and such signals would have  
 29367 arrived after the *fork()*; see also `<signal.h>`.

29368 One approach that has been suggested to address the problem of signal inheritance across *fork()*  
 29369 is to add an [EINTR] error, which would be returned when a signal is detected during the call.  
 29370 While this is preferable to losing signals, it was not considered an optimal solution. Although it  
 29371 is not recommended for this purpose, such an error would be an allowable extension for an  
 29372 implementation.

29373 The [ENOMEM] error value is reserved for those implementations that detect and distinguish  
 29374 such a condition. This condition occurs when an implementation detects that there is not enough  
 29375 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate  
 29376 because there can never be enough memory (either primary or secondary storage) to perform  
 29377 the operation. Since *fork()* duplicates an existing process, this must be a condition where there is  
 29378 sufficient memory for one such process, but not for two. Many historical implementations  
 29379 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally  
 29380 distinct from [EAGAIN] from the perspective of a conforming application.

29381 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it  
 29382 and it should be reserved for the error condition specified there. The condition is not applicable  
 29383 on many implementations.

29384 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.  
 29385 A system that single-threads processes was clearly not intended and is considered an  
 29386 unacceptable "toy implementation" of this volume of POSIX.1-2008. The only objection  
 29387 anticipated to the phrase "executing independently" is testability, but this assertion should be

**fork()**

29388 testable. Such tests require that both the parent and child can block on a detectable action of the  
 29389 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be  
 29390 possible for the system to conform to the intent of this volume of POSIX.1-2008.

29391 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it  
 29392 occurs or not is not in any practical sense under the control of the application because the  
 29393 condition is usually a consequence of the user's use of the system, not of the application's code.  
 29394 Thus, no application can or should rely upon its occurrence under any circumstances, nor  
 29395 should the exact semantics of what concept of "user" is used be of concern to the application  
 29396 developer. Validation writers should be cognizant of this limitation.

29397 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread  
 29398 of control within the same program (which was originally only possible in POSIX by creating a  
 29399 new process); the other is to create a new process running a different program. In the latter case,  
 29400 the call to *fork()* is soon followed by a call to one of the *exec* functions.

29401 The general problem with making *fork()* work in a multi-threaded world is what to do with all  
 29402 of the threads. There are two alternatives. One is to copy all of the threads into the new process.  
 29403 This causes the programmer or implementation to deal with threads that are suspended on  
 29404 system calls or that might be about to execute system calls that should not be executed in the  
 29405 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the  
 29406 difficulty that the state of process-local resources is usually held in process memory. If a thread  
 29407 that is not calling *fork()* holds a resource, that resource is never released in the child process  
 29408 because the thread whose job it is to release the resource does not exist in the child process.

29409 When a programmer is writing a multi-threaded program, the first described use of *fork()*,  
 29410 creating new threads in the same program, is provided by the *pthread\_create()* function. The  
 29411 *fork()* function is thus used only to run new programs, and the effects of calling functions that  
 29412 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

29413 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*  
 29414 function lets all the threads in the parent be duplicated in the child. This essentially duplicates  
 29415 the state of the parent in the child. This allows threads in the child to continue processing and  
 29416 allows locks and the state to be preserved without explicit *pthread\_atfork()* code. The calling  
 29417 process has to ensure that the threads processing state that is shared between the parent and  
 29418 child (that is, file descriptors or MAP\_SHARED memory) behaves properly after *forkall()*. For  
 29419 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two  
 29420 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.  
 29421 If this is not desired behavior, the parent process has to synchronize with such threads before  
 29422 calling *forkall()*.

29423 While the *fork()* function is *async-signal-safe*, there is no way for an implementation to  
 29424 determine whether the fork handlers established by *pthread\_atfork()* are *async-signal-safe*. The  
 29425 fork handlers may attempt to execute portions of the implementation that are not *async-signal-*  
 29426 *safe*, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore  
 29427 undefined for the fork handlers to execute functions that are not *async-signal-safe* when *fork()* is  
 29428 called from a signal handler.

29429 When *forkall()* is called, threads, other than the calling thread, that are in functions that can  
 29430 return with an [EINTR] error may have those functions return [EINTR] if the implementation  
 29431 cannot ensure that the function behaves correctly in the parent and child. In particular,  
 29432 *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* need to return in order to ensure that the  
 29433 condition has not changed. These functions can be awakened by a spurious condition wakeup  
 29434 rather than returning [EINTR].

29435 **FUTURE DIRECTIONS**

29436 None.

29437 **SEE ALSO**29438 *alarm()*, *exec*, *fcntl()*, *posix\_trace\_attr\_getinherited()*, *posix\_trace\_eventid\_equal()*, *pthread\_atfork()*,  
29439 *semop()*, *signal()*, *times()*29440 XBD Section 4.11 (on page 110), **<sys/types.h>**, **<unistd.h>**29441 **CHANGE HISTORY**

29442 First released in Issue 1. Derived from Issue 1 of the SVID.

29443 **Issue 5**29444 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX  
29445 Threads Extension.29446 **Issue 6**29447 The following new requirements on POSIX implementations derive from alignment with the  
29448 Single UNIX Specification:

- 29449
- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
29450 required for conforming implementations of previous POSIX specifications, it was not  
29451 required for UNIX applications.

29452 The following changes were made to align with the IEEE P1003.1a draft standard:

- 29453
- The effect of *fork()* on a pending alarm call in the child process is clarified.

29454 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

29455 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

29456 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the  
29457 DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread\_atfork()*  
29458 function and async-signal safety.29459 **Issue 7**29460 Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous  
29461 input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.29462 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers,  
29463 and Threads options is moved to the Base.

29464 Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

**fpathconf()**29465 **NAME**29466 `fpathconf`, `pathconf` — get configurable pathname variables29467 **SYNOPSIS**29468 `#include <unistd.h>`29469 `long fpathconf(int fildes, int name);`29470 `long pathconf(const char *path, int name);`29471 **DESCRIPTION**29472 The `fpathconf()` and `pathconf()` functions shall determine the current value of a configurable limit  
29473 or option (*variable*) that is associated with a file or directory.29474 For `pathconf()`, the *path* argument points to the pathname of a file or directory.29475 For `fpathconf()`, the *filde*s argument is an open file descriptor.29476 The *name* argument represents the variable to be queried relative to that file or directory.  
29477 Implementations shall support all of the variables listed in the following table and may support  
29478 others. The variables in the following table come from `<limits.h>` or `<unistd.h>` and the  
29479 symbolic constants, defined in `<unistd.h>`, are the corresponding values used for *name*.

| Variable                    | Value of <i>name</i>     | Requirements |
|-----------------------------|--------------------------|--------------|
| {FILESIZEBITS}              | _PC_FILESIZEBITS         | 3,4          |
| {LINK_MAX}                  | _PC_LINK_MAX             | 1            |
| {MAX_CANON}                 | _PC_MAX_CANON            | 2            |
| {MAX_INPUT}                 | _PC_MAX_INPUT            | 2            |
| {NAME_MAX}                  | _PC_NAME_MAX             | 3,4          |
| {PATH_MAX}                  | _PC_PATH_MAX             | 4,5          |
| {PIPE_BUF}                  | _PC_PIPE_BUF             | 6            |
| {POSIX2_SYMLINKS}           | _PC_2_SYMLINKS           | 4            |
| {POSIX_ALLOC_SIZE_MIN}      | _PC_ALLOC_SIZE_MIN       | 10           |
| {POSIX_REC_INCR_XFER_SIZE}  | _PC_REC_INCR_XFER_SIZE   | 10           |
| {POSIX_REC_MAX_XFER_SIZE}   | _PC_REC_MAX_XFER_SIZE    | 10           |
| {POSIX_REC_MIN_XFER_SIZE}   | _PC_REC_MIN_XFER_SIZE    | 10           |
| {POSIX_REC_XFER_ALIGN}      | _PC_REC_XFER_ALIGN       | 10           |
| {SYMLINK_MAX}               | _PC_SYMLINK_MAX          | 4,9          |
| _POSIX_CHOWN_RESTRICTED     | _PC_CHOWN_RESTRICTED     | 7            |
| _POSIX_NO_TRUNC             | _PC_NO_TRUNC             | 3,4          |
| _POSIX_VDISABLE             | _PC_VDISABLE             | 2            |
| _POSIX_ASYNC_IO             | _PC_ASYNC_IO             | 8            |
| _POSIX_PRIO_IO              | _PC_PRIO_IO              | 8            |
| _POSIX_SYNC_IO              | _PC_SYNC_IO              | 8            |
| _POSIX_TIMESTAMP_RESOLUTION | _PC_TIMESTAMP_RESOLUTION | 1            |

29502 **Requirements**

- 29503 1. If *path* or *filde*s refers to a directory, the value returned shall apply to the directory itself.
- 29504 2. If *path* or *filde*s does not refer to a terminal file, it is unspecified whether an  
29505 implementation supports an association of the variable name with the specified file.
- 29506 3. If *path* or *filde*s refers to a directory, the value returned shall apply to filenames within the  
29507 directory.

- 29508 4. If *path* or *fildev* does not refer to a directory, it is unspecified whether an implementation  
29509 supports an association of the variable name with the specified file.
- 29510 5. If *path* or *fildev* refers to a directory, the value returned shall be the maximum length of a  
29511 relative pathname when the specified directory is the working directory.
- 29512 6. If *path* refers to a FIFO, or *fildev* refers to a pipe or FIFO, the value returned shall apply to  
29513 the referenced object. If *path* or *fildev* refers to a directory, the value returned shall apply to  
29514 any FIFO that exists or can be created within the directory. If *path* or *fildev* refers to any  
29515 other type of file, it is unspecified whether an implementation supports an association of  
29516 the variable name with the specified file.
- 29517 7. If *path* or *fildev* refers to a directory, the value returned shall apply to any files, other than  
29518 directories, that exist or can be created within the directory.
- 29519 8. If *path* or *fildev* refers to a directory, it is unspecified whether an implementation supports  
29520 an association of the variable name with the specified file.
- 29521 9. If *path* or *fildev* refers to a directory, the value returned shall be the maximum length of the  
29522 string that a symbolic link in that directory can contain.
- 29523 10. If *path* or *fildev* does not refer to a regular file, it is unspecified whether an  
29524 implementation supports an association of the variable name with the specified file. If an  
29525 implementation supports such an association for other than a regular file, the value  
29526 returned is unspecified.

**RETURN VALUE**

29527 If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return  $-1$  and set *errno* to  
29528 indicate the error.  
29529

29530 If the variable corresponding to *name* is described in `<limits.h>` as a maximum or minimum  
29531 value and the variable has no limit for the path or file descriptor, both *pathconf()* and *fpathconf()*  
29532 shall return  $-1$  without changing *errno*. Note that indefinite limits do not imply infinite limits;  
29533 see `<limits.h>`.

29534 If the implementation needs to use *path* to determine the value of *name* and the implementation  
29535 does not support the association of *name* with the file specified by *path*, or if the process did not  
29536 have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf()*  
29537 shall return  $-1$  and set *errno* to indicate the error.

29538 If the implementation needs to use *fildev* to determine the value of *name* and the implementation  
29539 does not support the association of *name* with the file specified by *fildev*, or if *fildev* is an invalid  
29540 file descriptor, *fpathconf()* shall return  $-1$  and set *errno* to indicate the error.

29541 Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or  
29542 directory without changing *errno*. The value returned shall not be more restrictive than the  
29543 corresponding value available to the application when it was compiled with the  
29544 implementation's `<limits.h>` or `<unistd.h>`.

29545 If the variable corresponding to *name* is dependent on an unsupported option, the results are  
29546 unspecified.

**ERRORS**

29547 The *pathconf()* function shall fail if:

29548 [EINVAL] The value of *name* is not valid.

**fpathconf()**

|       |                          |                                                                                                                                                                                                                                                                                                                                                |
|-------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29550 | [ELOOP]                  | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                                                                                                                                                                                                                                                     |
| 29551 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29552 | [EOVERFLOW]              | The value of <i>name</i> is <code>_PC_TIMESTAMP_RESOLUTION</code> and the resolution is larger than <code>{LONG_MAX}</code> .                                                                                                                                                                                                                  |
| 29553 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29554 |                          | The <i>fpathconf()</i> function may fail if:                                                                                                                                                                                                                                                                                                   |
| 29555 | [EACCES]                 | Search permission is denied for a component of the path prefix.                                                                                                                                                                                                                                                                                |
| 29556 | [EINVAL]                 | The implementation does not support an association of the variable <i>name</i> with the specified file.                                                                                                                                                                                                                                        |
| 29557 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29558 | [ELOOP]                  | More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>path</i> argument.                                                                                                                                                                                                                            |
| 29559 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29560 | [ENAMETOOLONG]           |                                                                                                                                                                                                                                                                                                                                                |
| 29561 |                          | The length of a component of a pathname is longer than <code>{NAME_MAX}</code> .                                                                                                                                                                                                                                                               |
| 29562 | [ENAMETOOLONG]           |                                                                                                                                                                                                                                                                                                                                                |
| 29563 |                          | The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .                                                                                                                                                      |
| 29564 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29565 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29566 | [ENOENT]                 | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.                                                                                                                                                                                                                                                   |
| 29567 | [ENOTDIR]                | A component of the path prefix is not a directory, or the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory. |
| 29568 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29569 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29570 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29571 |                          | The <i>fpathconf()</i> function shall fail if:                                                                                                                                                                                                                                                                                                 |
| 29572 | [EINVAL]                 | The value of <i>name</i> is not valid.                                                                                                                                                                                                                                                                                                         |
| 29573 | [EOVERFLOW]              | The value of <i>name</i> is <code>_PC_TIMESTAMP_RESOLUTION</code> and the resolution is larger than <code>{LONG_MAX}</code> .                                                                                                                                                                                                                  |
| 29574 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29575 |                          | The <i>fpathconf()</i> function may fail if:                                                                                                                                                                                                                                                                                                   |
| 29576 | [EBADF]                  | The <i>fdes</i> argument is not a valid file descriptor.                                                                                                                                                                                                                                                                                       |
| 29577 | [EINVAL]                 | The implementation does not support an association of the variable <i>name</i> with the specified file.                                                                                                                                                                                                                                        |
| 29578 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29579 | <b>EXAMPLES</b>          |                                                                                                                                                                                                                                                                                                                                                |
| 29580 |                          | None.                                                                                                                                                                                                                                                                                                                                          |
| 29581 | <b>APPLICATION USAGE</b> |                                                                                                                                                                                                                                                                                                                                                |
| 29582 |                          | Application developers should check whether an option, such as <code>_POSIX_ADVISORY_INFO</code> , is supported prior to obtaining and using values for related variables such as <code>{POSIX_ALLOC_SIZE_MIN}</code> .                                                                                                                        |
| 29583 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29584 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29585 | <b>RATIONALE</b>         |                                                                                                                                                                                                                                                                                                                                                |
| 29586 |                          | The <i>fpathconf()</i> function was proposed immediately after the <i>sysconf()</i> function when it was realized that some configurable values may differ across file system, directory, or device boundaries.                                                                                                                                |
| 29587 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29588 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29589 |                          | For example, <code>{NAME_MAX}</code> frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all pathname components to 14 bytes, as this                                               |
| 29590 |                          |                                                                                                                                                                                                                                                                                                                                                |
| 29591 |                          |                                                                                                                                                                                                                                                                                                                                                |

29592 would be the value specified in `<limits.h>` on such a system.

29593 Therefore, various useful values can be queried on any pathname or file descriptor, assuming  
29594 that appropriate privileges are in place.

29595 The value returned for the variable `{PATH_MAX}` indicates the longest relative pathname that  
29596 could be given if the specified directory is the current working directory of the process. A  
29597 process may not always be able to generate a name that long and use it if a subdirectory in the  
29598 pathname crosses into a more restrictive file system. Note that implementations are allowed to  
29599 accept pathnames longer than `{PATH_MAX}` bytes long, but are not allowed to return  
29600 pathnames longer than this unless the user specifies a larger buffer using a function that  
29601 provides a buffer size argument.

29602 The value returned for the variable `_POSIX_CHOWN_RESTRICTED` also applies to directories  
29603 that do not have file systems mounted on them. The value may change when crossing a mount  
29604 point, so applications that need to know should check for each directory. (An even easier check  
29605 is to try the `chown()` function and look for an error in case it happens.)

29606 Unlike the values returned by `sysconf()`, the pathname-oriented variables are potentially more  
29607 volatile and are not guaranteed to remain constant throughout the lifetime of the process. For  
29608 example, in between two calls to `pathconf()`, the file system in question may have been  
29609 unmounted and remounted with different characteristics.

29610 Also note that most of the errors are optional. If one of the variables always has the same value  
29611 on an implementation, the implementation need not look at `path` or `fildev` to return that value and  
29612 is, therefore, not required to detect any of the errors except the meaning of `[EINVAL]` that  
29613 indicates that the value of `name` is not valid for that variable.

29614 If the value of any of the limits is unspecified (logically infinite), they will not be defined in  
29615 `<limits.h>` and the `pathconf()` and `fpathconf()` functions return `-1` without changing `errno`. This  
29616 can be distinguished from the case of giving an unrecognized `name` argument because `errno` is set  
29617 to `[EINVAL]` in this case.

29618 Since `-1` is a valid return value for the `pathconf()` and `fpathconf()` functions, applications should  
29619 set `errno` to zero before calling them and check `errno` only if the return value is `-1`.

29620 For the case of `{SYMLINK_MAX}`, since both `pathconf()` and `open()` follow symbolic links, there  
29621 is no way that `path` or `fildev` could refer to a symbolic link.

29622 It was the intention of IEEE Std 1003.1d-1999 that the following variables:

29623 `{POSIX_ALLOC_SIZE_MIN}`  
29624 `{POSIX_REC_INCR_XFER_SIZE}`  
29625 `{POSIX_REC_MAX_XFER_SIZE}`  
29626 `{POSIX_REC_MIN_XFER_SIZE}`  
29627 `{POSIX_REC_XFER_ALIGN}`

29628 only applied to regular files, but Note 10 also permits implementation of the advisory semantics  
29629 on other file types unique to an implementation (for example, a character special device).

29630 The `[EOVERFLOW]` error for `_PC_TIMESTAMP_RESOLUTION` cannot occur on POSIX-  
29631 compliant file systems because POSIX requires a timestamp resolution no larger than one  
29632 second. Even on 32-bit systems, this can be represented without overflow.

**fpathconf()**29633 **FUTURE DIRECTIONS**

29634 None.

29635 **SEE ALSO**29636 *chown()*, *confstr()*, *sysconf()*29637 XBD *<limits.h>*, *<unistd.h>*29638 XCU *getconf*29639 **CHANGE HISTORY**

29640 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

29641 **Issue 5**

29642 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

29643 Large File Summit extensions are added.

29644 **Issue 6**29645 The following new requirements on POSIX implementations derive from alignment with the  
29646 Single UNIX Specification:

- 29647 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 29648 • The [ELOOP] mandatory error condition is added.
- 29649 • A second [ENAMETOOLONG] is added as an optional error condition.

29650 The following changes were made to align with the IEEE P1003.1a draft standard:

- 29651 • The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.

29652 The following *pathconf()* variables and their associated names are added for alignment with  
29653 IEEE Std 1003.1d-1999:

```
29654 {POSIX_ALLOC_SIZE_MIN}
29655 {POSIX_REC_INCR_XFER_SIZE}
29656 {POSIX_REC_MAX_XFER_SIZE}
29657 {POSIX_REC_MIN_XFER_SIZE}
29658 {POSIX_REC_XFER_ALIGN}
```

29659 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth  
29660 paragraph of the DESCRIPTION and removing shading and margin markers from the table.  
29661 This change is needed since implementations are required to support all of these symbols.

29662 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for  
29663 POSIX2\_SYMLINKS in the DESCRIPTION.

29664 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the  
29665 DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

```
29666 {POSIX_ALLOC_SIZE_MIN}
29667 {POSIX_REC_INCR_XFER_SIZE}
29668 {POSIX_REC_MAX_XFER_SIZE}
29669 {POSIX_REC_MIN_XFER_SIZE}
29670 {POSIX_REC_XFER_ALIGN}
```

29671 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN  
29672 VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable  
29673 is dependent on an unsupported option, and advising application developers to check for

29674 supported options prior to obtaining and using such values.

29675 **Issue 7**

29676 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

29677 Changes are made related to support for finegrained timestamps.

29678 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
29679 pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fpclassify()**

System Interfaces

29680 **NAME**

29681 fpclassify — classify real floating type

29682 **SYNOPSIS**

29683 #include &lt;math.h&gt;

29684 int fpclassify(real-floating x);

29685 **DESCRIPTION**

29686 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 29687 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29688 volume of POSIX.1-2008 defers to the ISO C standard.

29689 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,  
 29690 zero, or into another implementation-defined category. First, an argument represented in a  
 29691 format wider than its semantic type is converted to its semantic type. Then classification is based  
 29692 on the type of the argument.

29693 **RETURN VALUE**

29694 The *fpclassify()* macro shall return the value of the number classification macro appropriate to  
 29695 the value of its argument.

29696 **ERRORS**

29697 No errors are defined.

29698 **EXAMPLES**

29699 None.

29700 **APPLICATION USAGE**

29701 None.

29702 **RATIONALE**

29703 None.

29704 **FUTURE DIRECTIONS**

29705 None.

29706 **SEE ALSO**29707 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

29708 XBD &lt;math.h&gt;

29709 **CHANGE HISTORY**

29710 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

29711 **NAME**29712 `dprintf, fprintf, printf, snprintf, sprintf` — print formatted output29713 **SYNOPSIS**29714 `#include <stdio.h>`

```

29715 CX int dprintf(int fildes, const char *restrict format, ...);
29716 int fprintf(FILE *restrict stream, const char *restrict format, ...);
29717 int printf(const char *restrict format, ...);
29718 int snprintf(char *restrict s, size_t n,
29719     const char *restrict format, ...);
29720 int sprintf(char *restrict s, const char *restrict format, ...);

```

29721 **DESCRIPTION**

29722 CX Excluding `dprintf()`: The functionality described on this reference page is aligned with the ISO C  
 29723 standard. Any conflict between the requirements described here and the ISO C standard is  
 29724 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

29725 The `fprintf()` function shall place output on the named output *stream*. The `printf()` function shall  
 29726 place output on the standard output stream *stdout*. The `sprintf()` function shall place output  
 29727 followed by the null byte, `'\0'`, in consecutive bytes starting at *s*; it is the user's responsibility  
 29728 to ensure that enough space is available.

29729 CX The `dprintf()` function shall be equivalent to the `fprintf()` function, except that `dprintf()` shall  
 29730 write output to the file associated with the file descriptor specified by the *fildes* argument rather  
 29731 than place output on a stream.

29732 The `snprintf()` function shall be equivalent to `sprintf()`, with the addition of the *n* argument  
 29733 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s*  
 29734 may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of  
 29735 being written to the array, and a null byte is written at the end of the bytes actually written into  
 29736 the array.

29737 If copying takes place between objects that overlap as a result of a call to `sprintf()` or `snprintf()`,  
 29738 the results are undefined.

29739 Each of these functions converts, formats, and prints its arguments under control of the *format*.  
 29740 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is  
 29741 composed of zero or more directives: *ordinary characters*, which are simply copied to the output  
 29742 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more  
 29743 arguments. The results are undefined if there are insufficient arguments for the *format*. If the  
 29744 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are  
 29745 otherwise ignored.

29746 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 29747 to the next unused argument. In this case, the conversion specifier character `%` (see below)  
 29748 is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range `[1, {NL_ARGMAX}]`,  
 29749 giving the position of the argument in the argument list. This feature provides for the definition  
 29750 of format strings that select arguments in an order appropriate to specific languages (see the  
 29751 EXAMPLES section).

29752 The *format* can contain either numbered argument conversion specifications (that is, `"%n$"` and  
 29753 `"*m$"`), or unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The  
 29754 only exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing  
 29755 numbered and unnumbered argument specifications in a *format* string are undefined. When  
 29756 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 29757 leading arguments, from the first to the (*N*-1)th, are specified in the format string.

**fprintf()**

- 29758 In format strings containing the "%n\$" form of conversion specification, numbered arguments  
29759 in the argument list can be referenced from the format string as many times as required.
- 29760 In format strings containing the % form of conversion specification, each conversion specification  
29761 uses the first unused argument in the argument list.
- 29762 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix  
29763 character in the output string. The radix character is defined in the process' locale (category  
29764 LC\_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the  
29765 radix character shall default to a <period> ('.').
- 29766 CX Each conversion specification is introduced by the '%' character or by the character sequence  
29767 "%n\$", after which the following appear in sequence:
- 29768 • Zero or more *flags* (in any order), which modify the meaning of the conversion  
29769 specification.
  - 29770 • An optional minimum *field width*. If the converted value has fewer bytes than the field  
29771 width, it shall be padded with <space> characters by default on the left; it shall be padded  
29772 on the right if the left-adjustment flag ('-'), described below, is given to the field width.  
29773 The field width takes the form of an <asterisk> ('\*'), described below, or a decimal  
29774 integer.
  - 29775 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,  
29776 x, and X conversion specifiers; the number of digits to appear after the radix character for  
29777 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for  
29778 the g and G conversion specifiers; or the maximum number of bytes to be printed from a  
29779 XSI string in the s and S conversion specifiers. The precision takes the form of a <period>  
29780 ('.') followed either by an <asterisk> ('\*'), described below, or an optional decimal digit  
29781 string, where a null digit string is treated as zero. If a precision appears with any other  
29782 conversion specifier, the behavior is undefined.
  - 29783 • An optional length modifier that specifies the size of the argument.
  - 29784 • A *conversion specifier* character that indicates the type of conversion to be applied.
- 29785 A field width, or precision, or both, may be indicated by an <asterisk> ('\*'). In this case an  
29786 argument of type *int* supplies the field width or precision. Applications shall ensure that  
29787 arguments specifying field width, or precision, or both appear in that order before the argument,  
29788 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field  
29789 CX width. A negative precision is taken as if the precision were omitted. In *format* strings  
29790 containing the "%n\$" form of a conversion specification, a field width or precision may be  
29791 indicated by the sequence "\*m\$", where *m* is a decimal integer in the range [1,{NL\_ARGMAX}]  
29792 giving the position in the argument list (after the *format* argument) of an integer argument  
29793 containing the field width or precision, for example:
- 29794 `printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);`
- 29795 The flag characters and their meanings are:
- 29796 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,  
29797 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For  
29798 other conversions the behavior is undefined. The non-monetary grouping character is  
29799 used.
- 29800 - The result of the conversion shall be left-justified within the field. The conversion is  
29801 right-justified if this flag is not specified.

|       |              |                                                                                                       |
|-------|--------------|-------------------------------------------------------------------------------------------------------|
| 29802 | +            | The result of a signed conversion shall always begin with a sign ('+' or '-'). The                    |
| 29803 |              | conversion shall begin with a sign only when a negative value is converted if this flag is            |
| 29804 |              | not specified.                                                                                        |
| 29805 | <space>      | If the first character of a signed conversion is not a sign or if a signed conversion results         |
| 29806 |              | in no characters, a <space> shall be prefixed to the result. This means that if the                   |
| 29807 |              | <space> and '+' flags both appear, the <space> flag shall be ignored.                                 |
| 29808 | #            | Specifies that the value is to be converted to an alternative form. For o conversion, it              |
| 29809 |              | increases the precision (if necessary) to force the first digit of the result to be zero. For x       |
| 29810 |              | or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A,         |
| 29811 |              | e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix                   |
| 29812 |              | character, even if no digits follow the radix character. Without this flag, a radix                   |
| 29813 |              | character appears in the result of these conversions only if a digit follows it. For g and G          |
| 29814 |              | conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they             |
| 29815 |              | normally are. For other conversion specifiers, the behavior is undefined.                             |
| 29816 | 0            | For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros                 |
| 29817 |              | (following any indication of sign or base) are used to pad to the field width rather than             |
| 29818 |              | performing space padding, except when converting an infinity or NaN. If the '0' and                   |
| 29819 |              | '-' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion                   |
| 29820 | CX           | specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and                |
| 29821 |              | <apostrophe> flags both appear, the grouping characters are inserted before zero                      |
| 29822 |              | padding. For other conversions, the behavior is undefined.                                            |
| 29823 |              | The length modifiers and their meanings are:                                                          |
| 29824 | hh           | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>signed char</b>   |
| 29825 |              | or <b>unsigned char</b> argument (the argument will have been promoted according to the               |
| 29826 |              | integer promotions, but its value shall be converted to <b>signed char</b> or <b>unsigned char</b>    |
| 29827 |              | before printing); or that a following n conversion specifier applies to a pointer to a                |
| 29828 |              | <b>signed char</b> argument.                                                                          |
| 29829 | h            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>short</b> or      |
| 29830 |              | <b>unsigned short</b> argument (the argument will have been promoted according to the                 |
| 29831 |              | integer promotions, but its value shall be converted to <b>short</b> or <b>unsigned short</b> before  |
| 29832 |              | printing); or that a following n conversion specifier applies to a pointer to a <b>short</b>          |
| 29833 |              | argument.                                                                                             |
| 29834 | l (ell)      | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long</b> or       |
| 29835 |              | <b>unsigned long</b> argument; that a following n conversion specifier applies to a pointer to        |
| 29836 |              | a <b>long</b> argument; that a following c conversion specifier applies to a <b>wint_t</b> argument;  |
| 29837 |              | that a following s conversion specifier applies to a pointer to a <b>wchar_t</b> argument; or         |
| 29838 |              | has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.                          |
| 29839 | ll (ell-ell) |                                                                                                       |
| 29840 |              | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long long</b>     |
| 29841 |              | or <b>unsigned long long</b> argument; or that a following n conversion specifier applies to a        |
| 29842 |              | pointer to a <b>long long</b> argument.                                                               |
| 29843 | j            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to an <b>intmax_t</b>     |
| 29844 |              | or <b>uintmax_t</b> argument; or that a following n conversion specifier applies to a pointer         |
| 29845 |              | to an <b>intmax_t</b> argument.                                                                       |
| 29846 | z            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>size_t</b> or the |
| 29847 |              | corresponding signed integer type argument; or that a following n conversion specifier                |
| 29848 |              | applies to a pointer to a signed integer type corresponding to a <b>size_t</b> argument.              |

**fprintf()**

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29849 | t                   | Specifies that a following <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , or <i>X</i> conversion specifier applies to a <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type argument; or that a following <i>n</i> conversion specifier applies to a pointer to a <b>ptrdiff_t</b> argument.                                                                                                                                                                                                                                              |
| 29850 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29851 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29852 | L                   | Specifies that a following <i>a</i> , <i>A</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , or <i>G</i> conversion specifier applies to a <b>long double</b> argument.                                                                                                                                                                                                                                                                                                                                                                       |
| 29853 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29854 |                     | If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29855 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29856 |                     | The conversion specifiers and their meanings are:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 29857 | <i>d</i> , <i>i</i> | The <b>int</b> argument shall be converted to a signed decimal in the style "[ <i>o</i> ] <i>ddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                   |
| 29858 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29859 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29860 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29861 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29862 | <i>o</i>            | The <b>unsigned</b> argument shall be converted to unsigned octal format in the style " <i>ddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                     |
| 29863 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29864 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29865 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29866 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29867 | <i>u</i>            | The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style " <i>ddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                   |
| 29868 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29869 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29870 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29871 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29872 | <i>x</i>            | The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style " <i>ddd</i> "; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                |
| 29873 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29874 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29875 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29876 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29877 | <i>X</i>            | Equivalent to the <i>x</i> conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 29878 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29879 | <i>f</i> , <i>F</i> | The <b>double</b> argument shall be converted to decimal notation in the style "[ <i>-</i> ] <i>ddd</i> . <i>ddd</i> ", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no ' <i>#</i> ' flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.                       |
| 29880 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29881 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29882 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29883 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29884 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29885 |                     | A <b>double</b> argument representing an infinity shall be converted in one of the styles "[ <i>-</i> ]inf" or "[ <i>-</i> ]infinity"; which style is implementation-defined. A <b>double</b> argument representing a NaN shall be converted in one of the styles "[ <i>-</i> ]nan( <i>n-char-sequence</i> )" or "[ <i>-</i> ]nan"; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The <i>F</i> conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively. |
| 29886 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29887 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29888 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29889 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29890 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29891 | <i>e</i> , <i>E</i> | The <b>double</b> argument shall be converted in the style "[ <i>-</i> ] <i>d</i> . <i>ddde</i> ± <i>dd</i> ", where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no ' <i>#</i> ' flag is present, no radix character shall                                                                                                                              |
| 29892 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29893 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29894 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|       |             |                                                                                                                                |
|-------|-------------|--------------------------------------------------------------------------------------------------------------------------------|
| 29895 |             | appear. The low-order digit shall be rounded in an implementation-defined manner.                                              |
| 29896 |             | The <b>E</b> conversion specifier shall produce a number with 'E' instead of 'e'                                               |
| 29897 |             | introducing the exponent. The exponent shall always contain at least two digits. If the                                        |
| 29898 |             | value is zero, the exponent shall be zero.                                                                                     |
| 29899 |             | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of                                    |
| 29900 |             | an <b>f</b> or <b>F</b> conversion specifier.                                                                                  |
| 29901 | <b>g, G</b> | The <b>double</b> argument representing a floating-point number shall be converted in the                                      |
| 29902 |             | style <b>f</b> or <b>e</b> (or in the style <b>F</b> or <b>E</b> in the case of a <b>G</b> conversion specifier), depending on |
| 29903 |             | the value converted and the precision. Let <b>P</b> equal the precision if non-zero, 6 if the                                  |
| 29904 |             | precision is omitted, or 1 if the precision is zero. Then, if a conversion with style <b>E</b>                                 |
| 29905 |             | would have an exponent of <b>X</b> :                                                                                           |
| 29906 |             | — If $P > X \geq -4$ , the conversion shall be with style <b>f</b> (or <b>F</b> ) and precision $P - (X + 1)$ .                |
| 29907 |             | — Otherwise, the conversion shall be with style <b>e</b> (or <b>E</b> ) and precision $P - 1$ .                                |
| 29908 |             | Finally, unless the '#' flag is used, any trailing zeros shall be removed from the                                             |
| 29909 |             | fractional portion of the result and the decimal-point character shall be removed if there                                     |
| 29910 |             | is no fractional portion remaining.                                                                                            |
| 29911 |             | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of                                    |
| 29912 |             | an <b>f</b> or <b>F</b> conversion specifier.                                                                                  |
| 29913 | <b>a, A</b> | A <b>double</b> argument representing a floating-point number shall be converted in the                                        |
| 29914 |             | style "[−]0xh.hhhhp±d", where there is one hexadecimal digit (which shall be non-                                              |
| 29915 |             | zero if the argument is a normalized floating-point number and is otherwise                                                    |
| 29916 |             | unspecified) before the decimal-point character and the number of hexadecimal digits                                           |
| 29917 |             | after it is equal to the precision; if the precision is missing and FLT_RADIX is a power                                       |
| 29918 |             | of 2, then the precision shall be sufficient for an exact representation of the value; if the                                  |
| 29919 |             | precision is missing and FLT_RADIX is not a power of 2, then the precision shall be                                            |
| 29920 |             | sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be                                     |
| 29921 |             | omitted; if the precision is zero and the '#' flag is not specified, no decimal-point                                          |
| 29922 |             | character shall appear. The letters "abcdef" shall be used for a conversion and the                                            |
| 29923 |             | letters "ABCDEF" for A conversion. The A conversion specifier produces a number with                                           |
| 29924 |             | 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one                                             |
| 29925 |             | digit, and only as many more digits as necessary to represent the decimal exponent of                                          |
| 29926 |             | 2. If the value is zero, the exponent shall be zero.                                                                           |
| 29927 |             | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of                                    |
| 29928 |             | an <b>f</b> or <b>F</b> conversion specifier.                                                                                  |
| 29929 | <b>c</b>    | The <b>int</b> argument shall be converted to an <b>unsigned char</b> , and the resulting byte shall                           |
| 29930 |             | be written.                                                                                                                    |
| 29931 |             | If an <b>l</b> (ell) qualifier is present, the <b>wint_t</b> argument shall be converted as if by an <b>ls</b>                 |
| 29932 |             | conversion specification with no precision and an argument that points to a two-                                               |
| 29933 |             | element array of type <b>wchar_t</b> , the first element of which contains the <b>wint_t</b> argument                          |
| 29934 |             | to the <b>ls</b> conversion specification and the second element contains a null wide                                          |
| 29935 |             | character.                                                                                                                     |
| 29936 | <b>s</b>    | The argument shall be a pointer to an array of <b>char</b> . Bytes from the array shall be                                     |
| 29937 |             | written up to (but not including) any terminating null byte. If the precision is specified,                                    |
| 29938 |             | no more than that many bytes shall be written. If the precision is not specified or is                                         |
| 29939 |             | greater than the size of the array, the application shall ensure that the array contains a                                     |
| 29940 |             | null byte.                                                                                                                     |

**fprintf()**

|       |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|-----|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29941 |     |                | If an <code>l</code> (ell) qualifier is present, the argument shall be a pointer to an array of type <code>wchar_t</code> . Wide characters from the array shall be converted to characters (each as if by a call to the <code>wcrtomb()</code> function, with the conversion state described by an <code>mbstate_t</code> object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written. |
| 29942 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29943 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29944 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29945 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29946 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29947 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29948 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29949 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29950 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29951 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29952 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29953 |     | <code>p</code> | The argument shall be a pointer to <code>void</code> . The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 29954 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29955 |     | <code>n</code> | The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the <code>fprintf()</code> functions. No argument is converted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29956 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29957 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29958 | XSI | <code>C</code> | Equivalent to <code>lc</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29959 | XSI | <code>S</code> | Equivalent to <code>ls</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29960 |     | <code>%</code> | Print a <code>' % '</code> character; no argument is converted. The complete conversion specification shall be <code>%%</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29961 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29962 |     |                | If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 29963 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29964 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29965 |     |                | In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <code>fprintf()</code> and <code>printf()</code> are printed as if <code>fputc()</code> had been called.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 29966 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29967 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29968 |     |                | For the <code>a</code> and <code>A</code> conversion specifiers, if <code>FLT_RADIX</code> is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29969 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29970 |     |                | For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 29971 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29972 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29973 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29974 |     |                | For the <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$ , both having <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string $D$ should satisfy $L \leq D \leq U$ , with the extra stipulation that the error should have a correct sign for the current rounding direction.                                                                                                                                                                                                               |
| 29975 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29976 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29977 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29978 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29979 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29980 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29981 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29982 | CX  |                | The last data modification and last file status change timestamps of the file shall be marked for update:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 29983 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29984 |     |                | 1. Between the call to a successful execution of <code>fprintf()</code> or <code>printf()</code> and the next successful completion of a call to <code>fflush()</code> or <code>fclose()</code> on the same stream or a call to <code>exit()</code> or <code>abort()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29985 |     |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

29986 2. Upon successful completion of a call to `dprintf()`

#### 29987 RETURN VALUE

29988 CX Upon successful completion, the `dprintf()`, `fprintf()`, and `printf()` functions shall return the  
29989 number of bytes transmitted.

29990 Upon successful completion, the `sprintf()` function shall return the number of bytes written to `s`  
29991 excluding the terminating null byte.

29992 Upon successful completion, the `snprintf()` function shall return the number of bytes that would  
29993 be written to `s` had `n` been sufficiently large excluding the terminating null byte.

29994 CX If an output error was encountered, these functions shall return a negative value and set `errno` to  
29995 indicate the error.

29996 If the value of `n` is zero on a call to `snprintf()`, nothing shall be written, the number of bytes that  
29997 would have been written had `n` been sufficiently large excluding the terminating null shall be  
29998 returned, and `s` may be a null pointer.

#### 29999 ERRORS

30000 CX For the conditions under which `dprintf()`, `fprintf()`, and `printf()` fail and may fail, refer to `fputc()`  
30001 or `fputwc()`.

30002 In addition, all forms of `fprintf()` shall fail if:

30003 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been  
30004 detected.

30005 In addition, all forms of `fprintf()` may fail if:

30006 CX [EINVAL] There are insufficient arguments.

30007 The `dprintf()` function may fail if:

30008 [EBADF] The `files` argument is not a valid file descriptor.

30009 CX The `dprintf()`, `fprintf()`, and `printf()` functions may fail if:

30010 CX [ENOMEM] Insufficient storage space is available.

30011 The `snprintf()` function shall fail if:

30012 CX [EOVERFLOW] The value of `n` is greater than `{INT_MAX}` or the number of bytes needed to  
30013 hold the output excluding the terminating null is greater than `{INT_MAX}`.

#### 30014 EXAMPLES

##### 30015 Printing Language-Independent Date and Time

30016 The following statement can be used to print date and time using a language-independent  
30017 format:

```
30018 printf(format, weekday, month, day, hour, min);
```

30019 For American usage, `format` could be a pointer to the following string:

```
30020 "%s, %s %d, %d:%.2d\n"
```

30021 This example would produce the following message:

```
30022 Sunday, July 3, 10:02
```

30023 For German usage, `format` could be a pointer to the following string:

**fprintf()**

30024 "%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n"

30025 This definition of *format* would produce the following message:

30026 Sonntag, 3. Juli, 10:02

### 30027 **Printing File Information**

30028 The following example prints information about the type, permissions, and number of links of a  
30029 specific file in a directory.

30030 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined  
30031 *strperm()* function shall return a string similar to the one at the beginning of the output for the  
30032 following command:

30033 `ls -l`

30034 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*  
30035 function shall return a **passwd** structure from which the name of the user is extracted. If the user  
30036 name is not found, the program instead prints out the numeric value of the user ID.

30037 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar  
30038 to *getpwuid()* except that it shall return group information based on the group number. Once  
30039 again, if the group is not found, the program prints the numeric value of the group for the entry.

30040 The final call to *printf()* prints the size of the file.

```
30041 #include <stdio.h>
30042 #include <sys/types.h>
30043 #include <pwd.h>
30044 #include <grp.h>
30045
30046 char *strperm (mode_t);
30047 ...
30048 struct stat statbuf;
30049 struct passwd *pwd;
30050 struct group *grp;
30051 ...
30052 printf("%10.10s", strperm (statbuf.st_mode));
30053 printf("%4d", statbuf.st_nlink);
30054
30055 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
30056     printf(" %-8.8s", pwd->pw_name);
30057 else
30058     printf(" %-8ld", (long) statbuf.st_uid);
30059
30060 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
30061     printf(" %-8.8s", grp->gr_name);
30062 else
30063     printf(" %-8ld", (long) statbuf.st_gid);
30064
30065 printf("%9jd", (intmax_t) statbuf.st_size);
30066 ...
```

30063 **Printing a Localized Date String**

30064 The following example gets a localized date string. The `nl_langinfo()` function shall return the  
 30065 localized date string, which specifies the order and layout of the date. The `strftime()` function  
 30066 takes this information and, using the `tm` structure for values, places the date and time  
 30067 information into `datestring`. The `printf()` function then outputs `datestring` and the name of the  
 30068 entry.

```
30069 #include <stdio.h>
30070 #include <time.h>
30071 #include <langinfo.h>
30072 ...
30073 struct dirent *dp;
30074 struct tm *tm;
30075 char datestring[256];
30076 ...
30077 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
30078 printf(" %s %s\n", datestring, dp->d_name);
30079 ...
```

30080 **Printing Error Information**

30081 The following example uses `fprintf()` to write error information to standard error.

30082 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If  
 30083 the file already exists, this is an error, as indicated by the `O_EXCL` flag on the `open()` function. If  
 30084 the call fails, the program assumes that someone else is updating the password file, and the  
 30085 program exits.

30086 The next group of calls saves a new password file as the current password file by creating a link  
 30087 between **LOCKFILE** and the new password file **PASSWDFILE**.

```
30088 #include <sys/types.h>
30089 #include <sys/stat.h>
30090 #include <fcntl.h>
30091 #include <stdio.h>
30092 #include <stdlib.h>
30093 #include <unistd.h>
30094 #include <string.h>
30095 #include <errno.h>
30096 #define LOCKFILE "/etc/ptmp"
30097 #define PASSWDFILE "/etc/passwd"
30098 ...
30099 int pfd;
30100 ...
30101 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
30102             S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
30103 {
30104     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
30105     exit(1);
30106 }
30107 ...
30108 if (link(LOCKFILE, PASSWDFILE) == -1) {
```

**fprintf()**

```

30109         fprintf(stderr, "Link error: %s\n", strerror(errno));
30110         exit(1);
30111     }
30112     ...

```

**Printing Usage Information**

The following example checks to make sure the program has the necessary arguments, and uses *fprintf()* to print usage information if the expected number of arguments is not present.

```

30116     #include <stdio.h>
30117     #include <stdlib.h>
30118     ...
30119     char *Options = "hdbt1";
30120     ...
30121     if (argc < 2) {
30122         fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
30123     }
30124     ...

```

**Formatting a Decimal String**

The following example prints a key and data pair on *stdout*. Note use of the <asterisk> ('\*') in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```

30129     #include <stdio.h>
30130     ...
30131     long i;
30132     char *keyst;
30133     int elementlen, len;
30134     ...
30135     while (len < elementlen) {
30136         ...
30137         printf("%s Element%0*ld\n", keyst, elementlen, i);
30138         ...
30139     }

```

**Creating a Filename**

The following example creates a filename using information from a previous *getpwnam()* function that returned the HOME directory of the user.

```

30143     #include <stdio.h>
30144     #include <sys/types.h>
30145     #include <unistd.h>
30146     ...
30147     char filename[PATH_MAX+1];
30148     struct passwd *pw;
30149     ...
30150     sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
30151     ...

```

30152 **Reporting an Event**

30153 The following example loops until an event has timed out. The *pause()* function waits forever  
 30154 unless it receives a signal. The *fprintf()* statement should never occur due to the possible return  
 30155 values of *pause()*.

```

30156 #include <stdio.h>
30157 #include <unistd.h>
30158 #include <string.h>
30159 #include <errno.h>
30160 ...
30161 while (!event_complete) {
30162     ...
30163     if (pause() != -1 || errno != EINTR)
30164         fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
30165 }
30166 ...

```

30167 **Printing Monetary Information**

30168 The following example uses *strfmon()* to convert a number and store it as a formatted monetary  
 30169 string named *convbuf*. If the first number is printed, the program prints the format and the  
 30170 description; otherwise, it just prints the number.

```

30171 #include <monetary.h>
30172 #include <stdio.h>
30173 ...
30174 struct tblfmt {
30175     char *format;
30176     char *description;
30177 };
30178 struct tblfmt table[] = {
30179     { "%n", "default formatting" },
30180     { "%11n", "right align within an 11 character field" },
30181     { "%#5n", "aligned columns for values up to 99 999" },
30182     { "%*#5n", "specify a fill character" },
30183     { "%=#5n", "fill characters do not use grouping" },
30184     { "%^#5n", "disable the grouping separator" },
30185     { "%^#5.0n", "round off to whole units" },
30186     { "%^#5.4n", "increase the precision" },
30187     { "%(#5n", "use an alternative pos/neg style" },
30188     { "%!(#5n", "disable the currency symbol" },
30189 };
30190 ...
30191 float input[3];
30192 int i, j;
30193 char convbuf[100];
30194 ...
30195 strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
30196
30197 if (j == 0) {
30198     printf("%s%s%s\n", table[i].format,
30199         convbuf, table[i].description);

```

**fprintf()**

```

30199     }
30200     else {
30201         printf("%s\n", convbuf);
30202     }
30203     ...

```

**30204 Printing Wide Characters**

30205 The following example prints a series of wide characters. Suppose that "L '@ '" expands to three  
 30206 bytes:

```

30207     wchar_t wz [3] = L"@@";           // Zero-terminated
30208     wchar_t wn [3] = L"@@";           // Unterminated

30209     fprintf (stdout, "%ls", wz);      // Outputs 6 bytes
30210     fprintf (stdout, "%ls", wn);      // Undefined because wn has no terminator
30211     fprintf (stdout, "%4ls", wz);     // Outputs 3 bytes
30212     fprintf (stdout, "%4ls", wn);     // Outputs 3 bytes; no terminator needed
30213     fprintf (stdout, "%9ls", wz);     // Outputs 6 bytes
30214     fprintf (stdout, "%9ls", wn);     // Outputs 9 bytes; no terminator needed
30215     fprintf (stdout, "%10ls", wz);    // Outputs 6 bytes
30216     fprintf (stdout, "%10ls", wn);    // Undefined because wn has no terminator

```

30217 In the last line of the example, after processing three characters, nine bytes have been output.  
 30218 The fourth character must then be examined to determine whether it converts to one byte or  
 30219 more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth  
 30220 character in the array, the behavior is undefined.

**30221 APPLICATION USAGE**

30222 If the application calling *fprintf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
 30223 the **<wchar.h>** header to have these objects defined.

**30224 RATIONALE**

30225 None.

**30226 FUTURE DIRECTIONS**

30227 None.

**30228 SEE ALSO**

30229 *fputc()*, *fscanf()*, *setlocale()*, *strfmon()*, *wcrtomb()*

30230 XBD Chapter 7 (on page 135), **<stdio.h>**, **<wchar.h>**

**30231 CHANGE HISTORY**

30232 First released in Issue 1. Derived from Issue 1 of the SVID.

**30233 Issue 5**

30234 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier can  
 30235 now be used with **c** and **s** conversion specifiers.

30236 The *snprintf()* function is new in Issue 5.

**30237 Issue 6**

30238 Extensions beyond the ISO C standard are marked.

30239 The normative text is updated to avoid use of the term "must" for application requirements.

- 30240 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 30241 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI shading is removed from *snprintf()*.
  - 30242
  - 30243 • The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed the behavior from Issue 5.
  - 30244
  - 30245
  - 30246 • The DESCRIPTION is updated.
- 30247 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.
- 30248
- 30249 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
- 30250 An example of printing wide characters is added.
- 30251 **Issue 7**
- 30252 Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.
- 30253
- 30254 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 30255 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of *g* and *G*.
- 30256
- 30257 SD5-XSH-ERN-174 is applied.
- 30258 The *dprintf()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
- 30259
- 30260 Functionality relating to the *%n\$* form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.
- 30261
- 30262 Changes are made related to support for finegrained timestamps.

**fputc()**30263 **NAME**30264 `fputc` — put a byte on a stream30265 **SYNOPSIS**30266 `#include <stdio.h>`30267 `int fputc(int c, FILE *stream);`30268 **DESCRIPTION**

30269 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30270 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30271 volume of POSIX.1-2008 defers to the ISO C standard.

30272 The `fputc()` function shall write the byte specified by `c` (converted to an **unsigned char**) to the  
 30273 output stream pointed to by `stream`, at the position indicated by the associated file-position  
 30274 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file  
 30275 cannot support positioning requests, or if the stream was opened with append mode, the byte  
 30276 shall be appended to the output stream.

30277 CX The last data modification and last file status change timestamps of the file shall be marked for  
 30278 update between the successful execution of `fputc()` and the next successful completion of a call  
 30279 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.

30280 **RETURN VALUE**

30281 Upon successful completion, `fputc()` shall return the value it has written. Otherwise, it shall  
 30282 CX return EOF, the error indicator for the stream shall be set, and `errno` shall be set to indicate the  
 30283 error.

30284 **ERRORS**

30285 The `fputc()` function shall fail if either the `stream` is unbuffered or the `stream`'s buffer needs to be  
 30286 flushed, and:

30287 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and  
 30288 the thread would be delayed in the write operation.

30289 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for  
 30290 writing.

30291 CX **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size.

30292 XSI **[EFBIG]** An attempt was made to write to a file that exceeds the file size limit of the  
 30293 process.

30294 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the  
 30295 offset maximum.

30296 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data  
 30297 was transferred.

30298 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background  
 30299 process group attempting to write to its controlling terminal, `TOSTOP` is set,  
 30300 the process is neither ignoring nor blocking `SIGTTOU`, and the process group  
 30301 of the process is orphaned. This error may also be returned under  
 30302 implementation-defined conditions.

30303 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

30304 CX **[EPIPE]** An attempt is made to write to a pipe or FIFO that is not open for reading by  
 30305 any process. A `SIGPIPE` signal shall also be sent to the thread.

30306 The *fputc()* function may fail if:

30307 CX [ENOMEM] Insufficient storage space is available.

30308 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.  
30309

#### 30310 EXAMPLES

30311 None.

#### 30312 APPLICATION USAGE

30313 None.

#### 30314 RATIONALE

30315 None.

#### 30316 FUTURE DIRECTIONS

30317 None.

#### 30318 SEE ALSO

30319 *error()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*

30320 XBD <stdio.h>

#### 30321 CHANGE HISTORY

30322 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 30323 Issue 5

30324 Large File Summit extensions are added.

#### 30325 Issue 6

30326 Extensions beyond the ISO C standard are marked.

30327 The following new requirements on POSIX implementations derive from alignment with the  
30328 Single UNIX Specification:

- 30329 • The [EIO] and [EFBIG] mandatory error conditions are added.
- 30330 • The [ENOMEM] and [ENXIO] optional error conditions are added.

30331 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN]  
30332 error in the ERRORS section from “the process would be delayed” to “the thread would be  
30333 delayed”.

#### 30334 Issue 7

30335 Changes are made related to support for finegrained timestamps.

**fputs()**30336 **NAME**

30337 fputs — put a string on a stream

30338 **SYNOPSIS**

30339 #include &lt;stdio.h&gt;

30340 int fputs(const char \*restrict s, FILE \*restrict stream);

30341 **DESCRIPTION**30342 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30343 conflict between the requirements described here and the ISO C standard is unintentional. This  
30344 volume of POSIX.1-2008 defers to the ISO C standard.30345 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed  
30346 to by *stream*. The terminating null byte shall not be written.30347 CX The last data modification and last file status change timestamps of the file shall be marked for  
30348 update between the successful execution of *fputs()* and the next successful completion of a call  
30349 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.30350 **RETURN VALUE**30351 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall  
30352 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.30353 **ERRORS**30354 Refer to *fputc()*.30355 **EXAMPLES**30356 **Printing to Standard Output**30357 The following example gets the current time, converts it to a string using *localtime()* and  
30358 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
30359 an event for which it is waiting.30360 #include <time.h>  
30361 #include <stdio.h>  
30362 ...  
30363 time\_t now;  
30364 int minutes\_to\_event;  
30365 ...  
30366 time(&now);  
30367 printf("The time is ");  
30368 fputs(asctime(localtime(&now)), stdout);  
30369 printf("There are still %d minutes to the event.\n",  
30370 minutes\_to\_event);  
3037130372 **APPLICATION USAGE**30373 The *puts()* function appends a <newline> while *fputs()* does not.30374 **RATIONALE**

30375 None.

30376 **FUTURE DIRECTIONS**

30377 None.

30378 **SEE ALSO**30379 *fopen()*, *putc()*, *puts()*30380 XBD `<stdio.h>`30381 **CHANGE HISTORY**

30382 First released in Issue 1. Derived from Issue 1 of the SVID.

30383 **Issue 6**

30384 Extensions beyond the ISO C standard are marked.

30385 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.30386 **Issue 7**

30387 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fputwc()**30388 **NAME**30389 `fputwc` — put a wide-character code on a stream30390 **SYNOPSIS**30391 `#include <stdio.h>`30392 `#include <wchar.h>`30393 `wint_t fputwc(wchar_t wc, FILE *stream);`30394 **DESCRIPTION**30395 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30396 conflict between the requirements described here and the ISO C standard is unintentional. This  
30397 volume of POSIX.1-2008 defers to the ISO C standard.30398 The `fputwc()` function shall write the character corresponding to the wide-character code `wc` to  
30399 the output stream pointed to by `stream`, at the position indicated by the associated file-position  
30400 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot  
30401 support positioning requests, or if the stream was opened with append mode, the character is  
30402 appended to the output stream. If an error occurs while writing the character, the shift state of  
30403 the output file is left in an undefined state.30404 CX The last data modification and last file status change timestamps of the file shall be marked for  
30405 update between the successful execution of `fputwc()` and the next successful completion of a call  
30406 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.30407 **RETURN VALUE**30408 Upon successful completion, `fputwc()` shall return `wc`. Otherwise, it shall return WEOF, the error  
30409 CX indicator for the stream shall be set, and `errno` shall be set to indicate the error.30410 **ERRORS**30411 The `fputwc()` function shall fail if either the stream is unbuffered or data in the `stream`'s buffer  
30412 needs to be written, and:30413 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and  
30414 the thread would be delayed in the write operation.30415 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for  
30416 writing.30417 CX **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size or  
30418 the file size limit of the process.30419 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the  
30420 offset maximum associated with the corresponding stream.30421 **[EILSEQ]** The wide-character code `wc` does not correspond to a valid character.30422 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data  
30423 was transferred.30424 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background  
30425 process group attempting to write to its controlling terminal, `TOSTOP` is set,  
30426 the process is neither ignoring nor blocking `SIGTTOU`, and the process group  
30427 of the process is orphaned. This error may also be returned under  
30428 implementation-defined conditions.30429 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

30430 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
30431 any process. A SIGPIPE signal shall also be sent to the thread.

30432 The *fputwc()* function may fail if:

30433 CX [ENOMEM] Insufficient storage space is available.

30434 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
30435 capabilities of the device.

#### 30436 EXAMPLES

30437 None.

#### 30438 APPLICATION USAGE

30439 None.

#### 30440 RATIONALE

30441 None.

#### 30442 FUTURE DIRECTIONS

30443 None.

#### 30444 SEE ALSO

30445 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*

30446 XBD `<stdio.h>`, `<wchar.h>`

#### 30447 CHANGE HISTORY

30448 First released in Issue 4. Derived from the MSE working draft.

#### 30449 Issue 5

30450 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
30451 is changed from `wint_t` to `wchar_t`.

30452 The Optional Header (OH) marking is removed from `<stdio.h>`.

30453 Large File Summit extensions are added.

#### 30454 Issue 6

30455 Extensions beyond the ISO C standard are marked.

30456 The following new requirements on POSIX implementations derive from alignment with the  
30457 Single UNIX Specification:

- 30458 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 30459 • The [ENOMEM] and [ENXIO] optional error conditions are added.

30460 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN]  
30461 error in the ERRORS section from “the process would be delayed” to “the thread would be  
30462 delayed”.

#### 30463 Issue 7

30464 Changes are made related to support for finegrained timestamps.

**fputws()**30465 **NAME**30466 `fputws` — put a wide-character string on a stream30467 **SYNOPSIS**30468 `#include <stdio.h>`30469 `#include <wchar.h>`30470 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`30471 **DESCRIPTION**30472 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30473 conflict between the requirements described here and the ISO C standard is unintentional. This  
30474 volume of POSIX.1-2008 defers to the ISO C standard.30475 The `fputws()` function shall write a character string corresponding to the (null-terminated) wide-  
30476 character string pointed to by `ws` to the stream pointed to by `stream`. No character corresponding  
30477 to the terminating null wide-character code shall be written.30478 CX The last data modification and last file status change timestamps of the file shall be marked for  
30479 update between the successful execution of `fputws()` and the next successful completion of a call  
30480 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.30481 **RETURN VALUE**30482 Upon successful completion, `fputws()` shall return a non-negative number. Otherwise, it shall  
30483 CX return `-1`, set an error indicator for the stream, and set `errno` to indicate the error.30484 **ERRORS**30485 Refer to `fputwc()`.30486 **EXAMPLES**

30487 None.

30488 **APPLICATION USAGE**30489 The `fputws()` function does not append a <newline>.30490 **RATIONALE**

30491 None.

30492 **FUTURE DIRECTIONS**

30493 None.

30494 **SEE ALSO**30495 `fopen()`30496 XBD `<stdio.h>`, `<wchar.h>`30497 **CHANGE HISTORY**

30498 First released in Issue 4. Derived from the MSE working draft.

30499 **Issue 5**30500 The Optional Header (OH) marking is removed from `<stdio.h>`.30501 **Issue 6**

30502 Extensions beyond the ISO C standard are marked.

30503 The `fputws()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.30504 **Issue 7**

30505 Changes are made related to support for finegrained timestamps.

30506 **NAME**

30507 fread — binary input

30508 **SYNOPSIS**

30509 #include &lt;stdio.h&gt;

30510 size\_t fread(void \*restrict ptr, size\_t size, size\_t nitems,  
30511 FILE \*restrict stream);30512 **DESCRIPTION**30513 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30514 conflict between the requirements described here and the ISO C standard is unintentional. This  
30515 volume of POSIX.1-2008 defers to the ISO C standard.30516 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size  
30517 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall  
30518 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**  
30519 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be  
30520 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the  
30521 file position indicator for the stream is unspecified. If a partial element is read, its value is  
30522 unspecified.30523 CX The *fread()* function may mark the last data access timestamp of the file associated with *stream*  
30524 for update. The last data access timestamp shall be marked for update by the first successful  
30525 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or  
30526 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.30527 **RETURN VALUE**30528 Upon successful completion, *fread()* shall return the number of elements successfully read which  
30529 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*  
30530 shall return 0 and the contents of the array and the state of the stream remain unchanged.30531 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall  
30532 be set to indicate the error.30533 **ERRORS**30534 Refer to *fgetc()*.30535 **EXAMPLES**30536 **Reading from a Stream**30537 The following example reads a single element from the *fp* stream into the array pointed to by  
30538 *buf*.30539 #include <stdio.h>  
30540 .  
30541 size\_t bytes\_read;  
30542 char buf[100];  
30543 FILE \*fp;  
30544 ...  
30545 bytes\_read = fread(buf, sizeof(buf), 1, fp);  
30546 ...30547 **APPLICATION USAGE**30548 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
30549 end-of-file condition.30550 Because of possible differences in element length and byte ordering, files written using *fwrite()*

**fread()**

30551 are application-dependent, and possibly cannot be read using *fread()* by a different application  
30552 or by the same application on a different processor.

30553 **RATIONALE**  
30554 None.

30555 **FUTURE DIRECTIONS**  
30556 None.

30557 **SEE ALSO**  
30558 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *fscanf()*, *getc()*, *gets()*  
30559 XBD `<stdio.h>`

30560 **CHANGE HISTORY**  
30561 First released in Issue 1. Derived from Issue 1 of the SVID.

30562 **Issue 6**  
30563 Extensions beyond the ISO C standard are marked.

30564 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 30565 • The *fread()* prototype is updated.
- 30566 • The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

30567 **Issue 7**  
30568 Changes are made related to support for finegrained timestamps.

30569 **NAME**

30570 free — free allocated memory

30571 **SYNOPSIS**

30572 #include &lt;stdlib.h&gt;

30573 void free(void \*ptr);

30574 **DESCRIPTION**

30575 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 30576 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30577 volume of POSIX.1-2008 defers to the ISO C standard.

30578 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made  
 30579 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the  
 30580 argument does not match a pointer earlier returned by a function in POSIX.1-2008 that allocates  
 30581 memory as if by *malloc()*, or if the space has been deallocated by a call to *free()* or *realloc()*, the  
 30582 behavior is undefined.

30583 Any use of a pointer that refers to freed space results in undefined behavior.

30584 **RETURN VALUE**30585 The *free()* function shall not return a value.30586 **ERRORS**

30587 No errors are defined.

30588 **EXAMPLES**

30589 None.

30590 **APPLICATION USAGE**

30591 There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

30592 **RATIONALE**

30593 None.

30594 **FUTURE DIRECTIONS**

30595 None.

30596 **SEE ALSO**30597 *calloc()*, *malloc()*, *posix\_memalign()*, *realloc()*

30598 XBD &lt;stdlib.h&gt;

30599 **CHANGE HISTORY**

30600 First released in Issue 1. Derived from Issue 1 of the SVID.

30601 **Issue 6**30602 Reference to the *valloc()* function is removed.30603 **Issue 7**

30604 The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that  
 30605 allocates memory as if by *malloc()*, then the behavior is undefined.

**freeaddrinfo()**

System Interfaces

30606 **NAME**

30607 freeaddrinfo, getaddrinfo — get address information

30608 **SYNOPSIS**

```

30609 #include <sys/socket.h>
30610 #include <netdb.h>

30611 void freeaddrinfo(struct addrinfo *ai);
30612 int getaddrinfo(const char *restrict nodename,
30613               const char *restrict servname,
30614               const struct addrinfo *restrict hints,
30615               struct addrinfo **restrict res);

```

30616 **DESCRIPTION**

30617 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,  
 30618 along with any additional storage associated with those structures. If the *ai\_next* field of the  
 30619 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall  
 30620 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

30621 The *getaddrinfo()* function shall translate the name of a service location (for example, a host  
 30622 name) and/or a service name and shall return a set of socket addresses and associated  
 30623 information to be used in creating a socket with which to address the specified service.

30624 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 30625 RFC 1035, and RFC 1886.

30626 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

30627 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated  
 30628 strings. One or both of these two arguments shall be supplied by the application as a non-null  
 30629 pointer.

30630 The format of a valid name depends on the address family or families. If a specific family is not  
 30631 given and the name could be interpreted as valid within multiple supported families, the  
 30632 implementation shall attempt to resolve the name in all supported families and, in absence of  
 30633 errors, one or more results shall be returned.

30634 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If  
 30635 IP6 the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, valid descriptive names  
 30636 include host names. If the specified address family is AF\_INET or AF\_UNSPEC, address strings  
 30637 using Internet standard dot notation as specified in *inet\_addr()* are valid.

30638 IP6 If the specified address family is AF\_INET6 or AF\_UNSPEC, standard IPv6 text forms described  
 30639 in *inet\_ntop()* are valid.

30640 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the  
 30641 requested service location is local to the caller.

30642 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If  
 30643 *servname* is not null, it is a null-terminated character string identifying the requested service.  
 30644 This can be either a descriptive name or a numeric representation suitable for use with the  
 30645 IP6 address family or families. If the specified address family is AF\_INET, AF\_INET6, or  
 30646 AF\_UNSPEC, the service can be specified as a string specifying a decimal port number.

30647 If the *hints* argument is not null, it refers to a structure containing input values that directs the  
 30648 operation by providing options and by limiting the returned information to a specific socket  
 30649 type, address family, and/or protocol, as described below. In this *hints* structure every member  
 30650 other than *ai\_flags*, *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be set to zero or a null pointer. A  
 30651 value of AF\_UNSPEC for *ai\_family* means that the caller shall accept any address family. A value

30652 of zero for *ai\_socktype* means that the caller shall accept any socket type. A value of zero for  
 30653 *ai\_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior  
 30654 shall be as if it referred to a structure containing the value zero for the *ai\_flags*, *ai\_socktype*, and  
 30655 *ai\_protocol* fields, and AF\_UNSPEC for the *ai\_family* field.

30656 The *ai\_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-  
 30657 inclusive OR of one or more of the values AI\_PASSIVE, AI\_CANONNAME,  
 30658 AI\_NUMERICHOST, AI\_NUMERICSERV, AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG.

30659 If the AI\_PASSIVE flag is specified, the returned address information shall be suitable for use in  
 30660 binding a socket for accepting incoming connections for the specified service. In this case, if the  
 30661 *nodename* argument is null, then the IP address portion of the socket address structure shall be  
 30662 set to INADDR\_ANY for an IPv4 address or IN6ADDR\_ANY\_INIT for an IPv6 address. If the  
 30663 AI\_PASSIVE flag is not specified, the returned address information shall be suitable for a call to  
 30664 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a  
 30665 connectionless protocol). In this case, if the *nodename* argument is null then the IP address  
 30666 portion of the socket address structure shall be set to the loopback address. The AI\_PASSIVE  
 30667 flag shall be ignored if the *nodename* argument is not null.

30668 If the AI\_CANONNAME flag is specified and the *nodename* argument is not null, the function  
 30669 shall attempt to determine the canonical name corresponding to *nodename* (for example, if  
 30670 *nodename* is an alias or shorthand notation for a complete name).

30671 **Note:** Since different implementations use different conceptual models, the terms “canonical name”  
 30672 and “alias” cannot be precisely defined for the general case. However, Domain Name System  
 30673 implementations are expected to interpret them as they are used in RFC 1034.

30674 A numeric host address string is not a “name”, and thus does not have a “canonical name”  
 30675 form; no address to host name translation is performed. See below for handling of the case  
 30676 where a canonical name cannot be obtained.

30677 If the AI\_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a  
 30678 numeric host address string. Otherwise, an [EAI\_NONAME] error is returned. This flag shall  
 30679 prevent any type of name resolution service (for example, the DNS) from being invoked.

30680 If the AI\_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a  
 30681 numeric port string. Otherwise, an [EAI\_NONAME] error shall be returned. This flag shall  
 30682 prevent any type of name resolution service (for example, NIS+) from being invoked.

30683 IP6 If the AI\_V4MAPPED flag is specified along with an *ai\_family* of AF\_INET6, then *getaddrinfo()*  
 30684 shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai\_addrlen*  
 30685 shall be 16). The AI\_V4MAPPED flag shall be ignored unless *ai\_family* equals AF\_INET6. If the  
 30686 AI\_ALL flag is used with the AI\_V4MAPPED flag, then *getaddrinfo()* shall return all matching  
 30687 IPv6 and IPv4 addresses. The AI\_ALL flag without the AI\_V4MAPPED flag is ignored.

30688 If the AI\_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4  
 30689 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6  
 30690 address is configured on the local system.

30691 The *ai\_socktype* field to which argument *hints* points specifies the socket type for the service, as  
 30692 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the  
 30693 service name could be interpreted as valid with multiple supported socket types, the  
 30694 implementation shall attempt to resolve the service name for all supported socket types and, in  
 30695 the absence of errors, all possible results shall be returned. A non-zero socket type value shall  
 30696 limit the returned information to values with the specified socket type.

30697 If the *ai\_family* field to which *hints* points has the value AF\_UNSPEC, addresses shall be  
 30698 returned for use with any address family that can be used with the specified *nodename* and/or

**freeaddrinfo()**

30700 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.  
 30701 If *ai\_family* is not AF\_UNSPEC and *ai\_protocol* is not zero, then addresses shall be returned for  
 30702 use only with the specified address family and protocol; the value of *ai\_protocol* shall be  
 30703 interpreted as in a call to the *socket()* function with the corresponding values of *ai\_family* and  
*ai\_protocol*.

30704 **RETURN VALUE**

30705 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value  
 30706 indicates failure. The possible values for the failures are listed in the ERRORS section.

30707 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list  
 30708 of **addrinfo** structures, each of which shall specify a socket address and information for use in  
 30709 creating a socket with which to use that socket address. The list shall include at least one  
 30710 **addrinfo** structure. The *ai\_next* field of each structure contains a pointer to the next structure on  
 30711 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall  
 30712 include values for use with a call to the *socket()* function, and a socket address for use with the  
 30713 *connect()* function or, if the AI\_PASSIVE flag was specified, for use with the *bind()* function. The  
 30714 fields *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be usable as the arguments to the *socket()*  
 30715 function to create a socket suitable for use with the returned address. The fields *ai\_addr* and  
 30716 *ai\_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,  
 30717 according to the AI\_PASSIVE flag.

30718 If *nodename* is not null, and if requested by the AI\_CANONNAME flag, the *ai\_canonname* field of  
 30719 the first returned **addrinfo** structure shall point to a null-terminated string containing the  
 30720 canonical name corresponding to the input *nodename*; if the canonical name is not available, then  
 30721 *ai\_canonname* shall refer to the *nodename* argument or a string with the same contents. The  
 30722 contents of the *ai\_flags* field of the returned structures are undefined.

30723 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an  
 30724 explicit argument (for example, *sin6\_flowinfo*) shall be set to zero.

30725 **Note:** This makes it easier to compare socket address structures.

30726 **ERRORS**

30727 The *getaddrinfo()* function shall fail and return the corresponding error value if:

30728 [EAL\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

30729 [EAL\_BADFLAGS] The *flags* parameter had an invalid value.  
 30730

30731 [EAL\_FAIL] A non-recoverable error occurred when attempting to resolve the name.

30732 [EAL\_FAMILY] The address family was not recognized.

30733 [EAL\_MEMORY] There was a memory allocation failure when trying to allocate storage for the  
 30734 return value.

30735 [EAL\_NONAME] The name does not resolve for the supplied parameters.

30736 Neither *nodename* nor *servname* were supplied. At least one of these shall be  
 30737 supplied.

30738 [EAL\_SERVICE] The service passed was not recognized for the specified socket type.

30739 [EAL\_SOCKETYPE] The intended socket type was not recognized.  
 30740

30741 [EAL\_SYSTEM] A system error occurred; the error code can be found in *errno*.

### 30742 EXAMPLES

30743 The following (incomplete) program demonstrates the use of *getaddrinfo()* to obtain the socket  
 30744 address structure(s) for the service named in the program's command-line argument. The  
 30745 program then loops through each of the address structures attempting to create and bind a  
 30746 socket to the address, until it performs a successful *bind()*.

```

30747 #include <stdio.h>
30748 #include <stdlib.h>
30749 #include <unistd.h>
30750 #include <string.h>
30751 #include <sys/socket.h>
30752 #include <netdb.h>

30753 int
30754 main(int argc, char *argv[])
30755 {
30756     struct addrinfo *result, *rp;
30757     int sfd, s;

30758     if (argc != 2) {
30759         fprintf(stderr, "Usage: %s port\n", argv[0]);
30760         exit(EXIT_FAILURE);
30761     }

30762     struct addrinfo hints = {};
30763     hints.ai_family = AF_UNSPEC;
30764     hints.ai_socktype = SOCK_DGRAM;
30765     hints.ai_flags = AI_PASSIVE;
30766     hints.ai_protocol = 0;

30767     s = getaddrinfo(NULL, argv[1], &hints, &result);
30768     if (s != 0) {
30769         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
30770         exit(EXIT_FAILURE);
30771     }

30772     /* getaddrinfo() returns a list of address structures.
30773     Try each address until a successful bind().
30774     If socket(2) (or bind(2)) fails, close the socket
30775     and try the next address. */

30776     for (rp = result; rp != NULL; rp = rp->ai_next) {
30777         sfd = socket(rp->ai_family, rp->ai_socktype,
30778             rp->ai_protocol);
30779         if (sfd == -1)
30780             continue;

30781         if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
30782             break; /* Success */

30783         close(sfd);
30784     }

30785     if (rp == NULL) { /* No address succeeded */
30786         fprintf(stderr, "Could not bind\n");

```

**freeaddrinfo()**

System Interfaces

```

30787         exit(EXIT_FAILURE);
30788     }
30789     freeaddrinfo(result);    /* No longer needed */
30790
30791     /* ... use socket bound to sfd ... */
30792 }

```

**APPLICATION USAGE**

30793 If the caller handles only TCP and not UDP, for example, then the *ai\_protocol* member of the *hints*  
 30794 structure should be set to IPPROTO\_TCP when *getaddrinfo()* is called.

30795 If the caller handles only IPv4 and not IPv6, then the *ai\_family* member of the *hints* structure  
 30796 should be set to AF\_INET when *getaddrinfo()* is called.

30797 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).  
 30798 It should be noted that the canonical name is a result of alias processing, and not necessarily a  
 30799 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this  
 30800 in the Domain Name System context.

**RATIONALE**

30801 None.  
 30802

**FUTURE DIRECTIONS**

30803 None.  
 30804

**SEE ALSO**

30805 *connect()*, *endservent()*, *gai\_strerror()*, *getnameinfo()*, *socket()*  
 30806

30807 XBD <netdb.h>, <sys/socket.h>

**CHANGE HISTORY**

30808 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

30810 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the  
 30811 ISO/IEC 9899:1999 standard.

30812 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the  
 30813 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical  
 30814 name”. A reference to RFC 2181 is also added to the Informative References.

30815 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for  
 30816 alignment with IPv6. These include the following:

- 30817 • Adding AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG to the allowed values for the  
 30818 *ai\_flags* field

- 30819 • Adding a description of AI\_ADDRCONFIG

- 30820 • Adding a description of the consequences of ignoring the AI\_PASSIVE flag.

30821 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing “corresponding  
 30822 value” to “corresponding error value” in the ERRORS section.

**Issue 7**

30824 Austin Group Interpretation 1003.1-2001 #013 is applied.

30825 Austin Group Interpretation 1003.1-2001 #146 is applied, updating the DESCRIPTION.

30826 An example is added.

30827 **NAME**30828            *freelocale* — free resources allocated for a locale object30829 **SYNOPSIS**

```
30830 CX      #include <locale.h>
30831          void freelocale(locale_t locobj);
```

30832 **DESCRIPTION**

30833        The *freelocale()* function shall cause the resources allocated for a locale object returned by a call  
 30834        to the *newlocale()* or *duplocale()* functions to be released.

30835        Any use of a locale object that has been freed results in undefined behavior.

30836 **RETURN VALUE**

30837            None.

30838 **ERRORS**

30839            None.

30840 **EXAMPLES**30841            **Freeing Up a Locale Object**

30842        The following example shows a code fragment to free a locale object created by *newlocale()*:

```
30843      #include <locale.h>
30844      ...
30845      /* Every locale object allocated with newlocale() should be
30846       * freed using freelocale():
30847       */
30848      locale_t loc;
30849      /* Get the locale. */
30850      loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
30851      /* ... Use the locale object ... */
30852      ...
30853      /* Free the locale object resources. */
30854      freelocale (loc);
```

30855 **APPLICATION USAGE**

30856            None.

30857 **RATIONALE**

30858            None.

30859 **FUTURE DIRECTIONS**

30860            None.

30861 **SEE ALSO**30862            *duplocale()*, *newlocale()*, *uselocale()*30863            XBD [<locale.h>](#)

**freelocale()**

|       |                            |
|-------|----------------------------|
| 30864 | <b>CHANGE HISTORY</b>      |
| 30865 | First released in Issue 7. |

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

30866 **NAME**

30867 freopen — open a stream

30868 **SYNOPSIS**

30869 #include &lt;stdio.h&gt;

30870 FILE \*freopen(const char \*restrict filename, const char \*restrict mode,  
30871 FILE \*restrict stream);30872 **DESCRIPTION**30873 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30874 conflict between the requirements described here and the ISO C standard is unintentional. This  
30875 volume of POSIX.1-2008 defers to the ISO C standard.30876 The *freopen()* function shall first attempt to flush the stream associated with *stream* as if by a call  
30877 to *fflush(stream)*. Failure to flush the stream successfully shall be ignored. If *filename* is not a null  
30878 pointer, *freopen()* shall close any file descriptor associated with *stream*. Failure to close the file  
30879 descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall  
30880 be cleared.30881 The *freopen()* function shall open the file whose pathname is the string pointed to by *filename*  
30882 and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in  
30883 *fopen()*.

30884 The original stream shall be closed regardless of whether the subsequent open succeeds.

30885 If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream  
30886 to that specified by *mode*, as if the name of the file currently associated with the stream had been  
30887 used. In this case, the file descriptor associated with the stream need not be closed if the call to  
30888 *freopen()* succeeds. It is implementation-defined which changes of mode are permitted (if any),  
30889 and under what circumstances.30890 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the  
30891 encoding rule shall be cleared, and the associated *mbstate\_t* object shall be set to describe an  
30892 initial conversion state.30893 CX If *filename* is not a null pointer, or if *filename* is a null pointer and the specified mode change  
30894 necessitates the file descriptor associated with the stream to be closed and reopened, the file  
30895 descriptor associated with the reopened stream shall be allocated and opened as if by a call to  
30896 *open()* with the following flags:

| <i>freopen()</i> Mode                 | <i>open()</i> Flags       |
|---------------------------------------|---------------------------|
| <i>r</i> or <i>rb</i>                 | O_RDONLY                  |
| <i>w</i> or <i>wb</i>                 | O_WRONLY O_CREAT O_TRUNC  |
| <i>a</i> or <i>ab</i>                 | O_WRONLY O_CREAT O_APPEND |
| <i>r+</i> or <i>rb+</i> or <i>r+b</i> | O_RDWR                    |
| <i>w+</i> or <i>wb+</i> or <i>w+b</i> | O_RDWR O_CREAT O_TRUNC    |
| <i>a+</i> or <i>ab+</i> or <i>a+b</i> | O_RDWR O_CREAT O_APPEND   |

30904 **RETURN VALUE**30905 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer  
30906 CX shall be returned, and *errno* shall be set to indicate the error.

**freopen()**30907 **ERRORS**30908 The *freopen()* function shall fail if:

- 30909 CX [EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.
- 30910  
30911  
30912
- 30913 CX [EBADF] The file descriptor underlying the stream is not a valid file descriptor when *filename* is a null pointer.
- 30914
- 30915 CX [EINTR] A signal was caught during *freopen()*.
- 30916 CX [EISDIR] The named file is a directory and *mode* requires write access.
- 30917 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.
- 30918
- 30919 CX [EMFILE] All file descriptors available to the process are currently open.
- 30920 CX [ENAMETOOLONG]
- 30921 The length of a component of a pathname is longer than {NAME\_MAX}.
- 30922 CX [ENFILE] The maximum allowable number of files is currently open in the system.
- 30923 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty string.
- 30924
- 30925 CX [ENOSPC] The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
- 30926
- 30927 CX [ENOTDIR] A component of the path prefix is not a directory, or the *filename* argument contains at least one non-`<slash>` character and ends with one or more trailing `<slash>` characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
- 30928  
30929  
30930
- 30931 CX [ENXIO] The named file is a character special or block special file, and the device associated with this special file does not exist.
- 30932
- 30933 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented correctly in an object of type `off_t`.
- 30934
- 30935 CX [EROFS] The named file resides on a read-only file system and *mode* requires write access.
- 30936

30937 The *freopen()* function may fail if:

- 30938 CX [EBADF] The mode with which the file descriptor underlying the stream was opened does not support the requested mode when *filename* is a null pointer.
- 30939
- 30940 CX [EINVAL] The value of the *mode* argument is not valid.
- 30941 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during resolution of the *path* argument.
- 30942
- 30943 CX [ENAMETOOLONG]
- 30944 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH\_MAX}.
- 30945  
30946

- 30947 CX [ENOMEM] Insufficient storage space is available.
- 30948 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.
- 30949
- 30950 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.
- 30951

## 30952 EXAMPLES

### 30953 Directing Standard Output to a File

30954 The following example logs all standard output to the `/tmp/logfile` file.

```
30955 #include <stdio.h>
30956 ...
30957 FILE *fp;
30958 ...
30959 fp = freopen ("/tmp/logfile", "a+", stdout);
30960 ...
```

## 30961 APPLICATION USAGE

30962 The `freopen()` function is typically used to attach the pre-opened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

30963

30964 Since implementations are not required to support any stream mode changes when the *filename* argument is `NULL`, portable applications cannot rely on the use of `freopen()` to change the stream mode, and use of this feature is discouraged. The feature was originally added to the ISO C standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a 'b' character in the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX applications. However, even though the 'b' is ignored, a successful call to `freopen(NULL, "wb", stdout)` does have an effect. In particular, for regular files it truncates the file and sets the file-position indicator for the stream to the start of the file. It is possible that these side-effects are an unintended consequence of the way the feature is specified in the ISO/IEC 9899:1999 standard, but unless or until the ISO C standard is changed, applications which successfully call `freopen(NULL, "wb", stdout)` will behave in unexpected ways on conforming systems in situations such as:

```
30976 { appl file1; appl file2; } > file3
```

30977 which will result in `file3` containing only the output from the second invocation of `appl`.

## 30978 RATIONALE

30979 None.

## 30980 FUTURE DIRECTIONS

30981 None.

## 30982 SEE ALSO

30983 `fclose()`, `fdopen()`, `fflush()`, `fmemopen()`, `fopen()`, `mbsinit()`, `open()`, `open_memstream()`

30984 XBD `<stdio.h>`

## 30985 CHANGE HISTORY

30986 First released in Issue 1. Derived from Issue 1 of the SVID.

**freopen()**30987 **Issue 5**

30988 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the  
 30989 conversion state of the stream is set to an initial conversion state by a successful call to the  
 30990 *freopen()* function.

30991 Large File Summit extensions are added.

30992 **Issue 6**

30993 Extensions beyond the ISO C standard are marked.

30994 The following new requirements on POSIX implementations derive from alignment with the  
 30995 Single UNIX Specification:

- 30996 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
 30997 file description. This change is to support large files.
- 30998 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
 30999 large files.
- 31000 • The [ELOOP] mandatory error condition is added.
- 31001 • A second [ENAMETOOLONG] is added as an optional error condition.
- 31002 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

31003 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 31004 • The *freopen()* prototype is updated.
- 31005 • The DESCRIPTION is updated.

31006 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 31007 [ELOOP] error condition is added.

31008 The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

31009 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following  
 31010 sentence to the DESCRIPTION: “In this case, the file descriptor associated with the stream need  
 31011 not be closed if the call to *freopen()* succeeds.”.

31012 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory  
 31013 [EBADF] error, and an optional [EBADF] error to the ERRORS section.

31014 **Issue 7**

31015 Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen()* function  
 31016 allocates a file descriptor as per *open()*.

31017 Austin Group Interpretation 1003.1-2001 #143 is applied.

31018 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set  
 31019 on the open file description.

31020 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

31021 SD5-XSH-ERN-150 and SD5-XSH-ERN-219 are applied.

31022 **NAME**

31023 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

31024 **SYNOPSIS**

31025 #include &lt;math.h&gt;

31026 double frexp(double num, int \*exp);

31027 float frexpf(float num, int \*exp);

31028 long double frexpl(long double num, int \*exp);

31029 **DESCRIPTION**31030 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
31031 conflict between the requirements described here and the ISO C standard is unintentional. This  
31032 volume of POSIX.1-2008 defers to the ISO C standard.31033 These functions shall break a floating-point number *num* into a normalized fraction and an  
31034 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.31035 **RETURN VALUE**31036 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the  
31037 interval  $[\frac{1}{2}, 1)$  or 0, and *num* equals *x* times 2 raised to the power *\*exp*.31038 **MX** If *num* is NaN, a NaN shall be returned, and the value of *\*exp* is unspecified.31039 If *num* is  $\pm 0$ ,  $\pm 0$  shall be returned, and the value of *\*exp* shall be 0.31040 If *num* is  $\pm \text{Inf}$ , *num* shall be returned, and the value of *\*exp* is unspecified.31041 **ERRORS**

31042 No errors are defined.

31043 **EXAMPLES**

31044 None.

31045 **APPLICATION USAGE**

31046 None.

31047 **RATIONALE**

31048 None.

31049 **FUTURE DIRECTIONS**

31050 None.

31051 **SEE ALSO**31052 *isnan()*, *ldexp()*, *modf()*

31053 XBD &lt;math.h&gt;

31054 **CHANGE HISTORY**

31055 First released in Issue 1. Derived from Issue 1 of the SVID.

31056 **Issue 5**31057 The DESCRIPTION is updated to indicate how an application should check for an error. This  
31058 text was previously published in the APPLICATION USAGE section.

**frexp()**31059 **Issue 6**

31060 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
31061 standard.

31062 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
31063 revised to align with the ISO/IEC 9899:1999 standard.

31064 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
31065 marked.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 31066 NAME

31067 fscanf, scanf, sscanf — convert formatted input

## 31068 SYNOPSIS

31069 #include &lt;stdio.h&gt;

31070 int fscanf(FILE \*restrict stream, const char \*restrict format, ...);

31071 int scanf(const char \*restrict format, ...);

31072 int sscanf(const char \*restrict s, const char \*restrict format, ...);

## 31073 DESCRIPTION

31074 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31075 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31076 volume of POSIX.1-2008 defers to the ISO C standard.

31077 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read  
 31078 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each  
 31079 function reads bytes, interprets them according to a format, and stores the results in its  
 31080 arguments. Each expects, as arguments, a control string *format* described below, and a set of  
 31081 *pointer* arguments indicating where the converted input should be stored. The result is  
 31082 undefined if there are insufficient arguments for the format. If the format is exhausted while  
 31083 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

31084 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 31085 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 31086 replaced by the sequence "%*n*\$", where *n* is a decimal integer in the range [1, {NL\_ARGMAX}].  
 31087 This feature provides for the definition of format strings that select arguments in an order  
 31088 appropriate to specific languages. In format strings containing the "%*n*\$" form of conversion  
 31089 specifications, it is unspecified whether numbered arguments in the argument list can be  
 31090 referenced from the format string more than once.

31091 The *format* can contain either form of a conversion specification—that is, % or "%*n*\$"—but the  
 31092 two forms cannot be mixed within a single *format* string. The only exception to this is that %% or  
 31093 %\* can be mixed with the "%*n*\$" form. When numbered argument specifications are used,  
 31094 specifying the *N*th argument requires that all the leading arguments, from the first to the  
 31095 (*N*−1)th, are pointers.

31096 The *fscanf()* function in all its forms shall allow detection of a language-dependent radix  
 31097 character in the input string. The radix character is defined in the locale of the process (category  
 31098 LC\_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the  
 31099 radix character shall default to a <period> ('.').

31100 The *format* is a character string, beginning and ending in its initial shift state, if any, composed  
 31101 of zero or more directives. Each directive is composed of one of the following: one or more  
 31102 white-space characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary  
 31103 character (neither '%' nor a white-space character); or a conversion specification. Each  
 31104 CX conversion specification is introduced by the character '%' or the character sequence "%*n*\$",  
 31105 after which the following appear in sequence:

- 31106 • An optional assignment-suppressing character '\*'.
- 31107 • An optional non-zero decimal integer that specifies the maximum field width.
- 31108 CX • An optional assignment-allocation character 'm'.
- 31109 • An option length modifier that specifies the size of the receiving object.

**fscanf()**

- 31110 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid  
31111 conversion specifiers are described below.
- 31112 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
31113 detailed below, the function shall return. Failures are described as input failures (due to the  
31114 unavailability of input bytes) or matching failures (due to inappropriate input).
- 31115 A directive composed of one or more white-space characters shall be executed by reading input  
31116 until no more valid input can be read, or up to the first byte which is not a white-space character,  
31117 which remains unread.
- 31118 A directive that is an ordinary character shall be executed as follows: the next byte shall be read  
31119 from the input and compared with the byte that comprises the directive; if the comparison  
31120 shows that they are not equivalent, the directive shall fail, and the differing and subsequent  
31121 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a  
31122 character from being read, the directive shall fail.
- 31123 A directive that is a conversion specification defines a set of matching input sequences, as  
31124 described below for each conversion character. A conversion specification shall be executed in  
31125 the following steps.
- 31126 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion  
31127 specification includes a `[`, `c`, `C`, or `n` conversion specifier.
- 31128 An item shall be read from the input, unless the conversion specification includes an `n`  
31129 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to  
31130 any specified maximum field width, which may be measured in characters or bytes dependent  
31131 on the conversion specifier) which is an initial subsequence of a matching sequence. The first  
31132 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the  
31133 execution of the conversion specification shall fail; this condition is a matching failure, unless  
31134 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
31135 an input failure.
- 31136 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
31137 specification, the count of input bytes) shall be converted to a type appropriate to the conversion  
31138 character. If the input item is not a matching sequence, the execution of the conversion  
31139 specification fails; this condition is a matching failure. Unless assignment suppression was  
31140 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the  
31141 first argument following the *format* argument that has not already received a conversion result if  
31142 CX the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the  
31143 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the  
31144 conversion cannot be represented in the space provided, the behavior is undefined.
- 31145 CX The `%c`, `%s`, and `%[` conversion specifiers shall accept an optional assignment-allocation  
31146 character `'m'`, which shall cause a memory buffer to be allocated to hold the string converted  
31147 including a terminating null character. In such a case, the argument corresponding to the  
31148 conversion specifier should be a reference to a pointer variable that will receive a pointer to the  
31149 allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application  
31150 shall be responsible for freeing the memory after usage. If there is insufficient memory to  
31151 allocate a buffer, the function shall set *errno* to `[ENOMEM]` and a conversion error shall result. If  
31152 the function returns EOF, any memory successfully allocated for parameters using assignment-  
31153 allocation character `'m'` by this call shall be freed before the function returns.

- 31154 The length modifiers and their meanings are:
- 31155 hh Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31156 argument with type pointer to **signed char** or **unsigned char**.
- 31157 h Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31158 argument with type pointer to **short** or **unsigned short**.
- 31159 l (ell) Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31160 argument with type pointer to **long** or **unsigned long**; that a following *a*, *A*, *e*, *E*, *f*, *F*,  
31161 *g*, or *G* conversion specifier applies to an argument with type pointer to **double**; or that  
31162 a following *c*, *s*, or *l* conversion specifier applies to an argument with type pointer to  
31163 **wchar\_t**. If the 'm' assignment-allocation character is specified, the conversion  
31164 applies to an argument with the type pointer to a pointer to **wchar\_t**.
- 31165 ll (ell-ell)  
31166 Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31167 argument with type pointer to **long long** or **unsigned long long**.
- 31168 j Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31169 argument with type pointer to **intmax\_t** or **uintmax\_t**.
- 31170 z Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31171 argument with type pointer to **size\_t** or the corresponding signed integer type.
- 31172 t Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
31173 argument with type pointer to **ptrdiff\_t** or the corresponding **unsigned** type.
- 31174 L Specifies that a following *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G* conversion specifier applies to an  
31175 argument with type pointer to **long double**.
- 31176 If a length modifier appears with any conversion specifier other than as specified above, the  
31177 behavior is undefined.
- 31178 The following conversion specifiers are valid:
- 31179 d Matches an optionally signed decimal integer, whose format is the same as expected for  
31180 the subject sequence of *strtol()* with the value 10 for the *base* argument. In the absence  
31181 of a size modifier, the application shall ensure that the corresponding argument is a  
31182 pointer to **int**.
- 31183 i Matches an optionally signed integer, whose format is the same as expected for the  
31184 subject sequence of *strtol()* with 0 for the *base* argument. In the absence of a size  
31185 modifier, the application shall ensure that the corresponding argument is a pointer to  
31186 **int**.
- 31187 o Matches an optionally signed octal integer, whose format is the same as expected for  
31188 the subject sequence of *strtoul()* with the value 8 for the *base* argument. In the absence  
31189 of a size modifier, the application shall ensure that the corresponding argument is a  
31190 pointer to **unsigned**.
- 31191 u Matches an optionally signed decimal integer, whose format is the same as expected for  
31192 the subject sequence of *strtoul()* with the value 10 for the *base* argument. In the absence  
31193 of a size modifier, the application shall ensure that the corresponding argument is a  
31194 pointer to **unsigned**.
- 31195 x Matches an optionally signed hexadecimal integer, whose format is the same as  
31196 expected for the subject sequence of *strtoul()* with the value 16 for the *base* argument. In  
31197 the absence of a size modifier, the application shall ensure that the corresponding

**fscanf()**

|       |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 31198 |            | argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 31199 | a, e, f, g |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31200 |            | Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod</i> ( ). In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 31201 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31202 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31203 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31204 |            | If the <i>fprintf</i> ( ) family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fscanf</i> ( ) family of functions shall recognize them as input.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 31205 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31206 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31207 | s          | Matches a sequence of bytes that are not white-space characters. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null character code, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                                                                                                                                                |
| 31208 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31209 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31210 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31211 | CX         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31212 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31213 |            | If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to the <i>mbrtowc</i> ( ) function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .                 |
| 31214 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31215 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31216 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31217 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31218 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31219 | CX         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31220 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31221 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31222 | [          | Matches a non-empty sequence of bytes from a set of expected bytes (the <i>scanset</i> ). The normal skip over white-space characters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null byte, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                                                   |
| 31223 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31224 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31225 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31226 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31227 | CX         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31228 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31229 |            | If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence shall be converted to a wide character as if by a call to the <i>mbrtowc</i> ( ) function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> . |
| 31230 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31231 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31232 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31233 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31234 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31235 | CX         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31236 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31237 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31238 |            | The conversion specification includes all subsequent bytes in the <i>format</i> string up to and including the matching <right-square-bracket> ( ' ] ' ). The bytes between the square brackets (the <i>scanlist</i> ) comprise the <i>scanset</i> , unless the byte after the <left-square-bracket> is a <circumflex> ( ' ^ ' ), in which case the <i>scanset</i> contains all bytes that do not appear in the <i>scanlist</i> between the <circumflex> and the <right-square-bracket>. If the conversion specification begins with " [ ] " or " [ ^ ] ", the <right-square-bracket> is included in the <i>scanlist</i> and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the                               |
| 31239 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31240 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31241 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31242 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31243 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31244 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31245 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|       |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 31246 |     | first <right-square-bracket> is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 31247 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31248 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31249 | c   | Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space characters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                             |
| 31250 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31251 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31252 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31253 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31254 | CX  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31255 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31256 |     | If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. No null wide character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> . |
| 31257 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31258 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31259 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31260 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31261 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31262 | CX  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31263 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31264 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31265 | p   | Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.                                                                                                                                                                  |
| 31266 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31267 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31268 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31269 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31270 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31271 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31272 | n   | No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the <i>fscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.                                                                                                                                                                                               |
| 31273 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31274 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31275 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31276 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31277 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31278 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31279 | XSI | C Equivalent to l c.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 31280 | XSI | S Equivalent to l s.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 31281 | %   | Matches a single '%' character; no conversion or assignment occurs. The complete conversion specification shall be %%.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 31282 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31283 |     | If a conversion specification is invalid, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 31284 |     | The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and x, respectively.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 31285 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31286 |     | If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.                                                                                                                                                                                                |
| 31287 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31288 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31289 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31290 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 31291 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**fscanf()**

31292 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for  
31293 *fscanf()*.

31294 If conversion terminates on a conflicting input, the offending input is left unread in the input.  
31295 Any trailing white space (including <newline> characters) shall be left unread unless matched  
31296 by a conversion specification. The success of literal matches and suppressed assignments is only  
31297 directly determinable via the %n conversion specification.

31298 CX The *fscanf()* and *scanf()* functions may mark the last data access timestamp of the file associated  
31299 with *stream* for update. The last data access timestamp shall be marked for update by the first  
31300 successful execution of *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*,  
31301 *fscanf()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

**RETURN VALUE**

31302 Upon successful completion, these functions shall return the number of successfully matched  
31303 and assigned input items; this number can be zero in the event of an early matching failure. If  
31304 the input ends before the first matching failure or conversion, EOF shall be returned. If any  
31305 error occurs, EOF shall be returned, and *errno* shall be set to indicate the error. If a read error  
31306 CX occurs, the error indicator for the stream shall be set.  
31307

**ERRORS**

31308 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or  
31309 *fgetwc()*.  
31310

31311 In addition, the *fscanf()* function shall fail if:

31312 CX [EILSEQ] Input byte sequence does not form a valid character.

31313 [ENOMEM] Insufficient storage space is available.

31314 In addition, the *fscanf()* function may fail if:

31315 CX [EINVAL] There are insufficient arguments.

**EXAMPLES**

31316 The call:

```
31317 int i, n; float x; char name[50];
31318 n = scanf("%d%f%s", &i, &x, name);
```

31319 with the input line:

```
31320 25 54.32E-1 Hamster
```

31321 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
31322 "Hamster".

31323 The call:

```
31324 int i; float x; char name[50];
31325 (void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

31326 with input:

```
31327 56789 0123 56a72
```

31328 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
31329 *getchar()* shall return the character 'a'.  
31330

31331 **Reading Data into an Array**

31332 The following call uses *fscanf()* to read three floating-point numbers from standard input into  
31333 the *input* array.

```
31334 float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

31335 **APPLICATION USAGE**

31336 If the application calling *fscanf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
31337 the **<wchar.h>** header to have these objects defined.

31338 For functions that allocate memory as if by *malloc()*, the application should release such memory  
31339 when it is no longer required by a call to *free()*. For *fscanf()*, this is memory allocated via use of  
31340 the 'm' assignment-allocation character.

31341 **RATIONALE**

31342 This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few "obvious"  
31343 things were not included. Specifically, the set of characters allowed in a scanset is limited to  
31344 single-byte characters. In other similar places, multi-byte characters have been permitted, but  
31345 for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications  
31346 needing this could use the corresponding wide-character functions to achieve the desired  
31347 results.

31348 **FUTURE DIRECTIONS**

31349 None.

31350 **SEE ALSO**

31351 *fprintf()*, *getc()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*

31352 XBD Chapter 7 (on page 135), **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

31353 **CHANGE HISTORY**

31354 First released in Issue 1. Derived from Issue 1 of the SVID.

31355 **Issue 5**

31356 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier is  
31357 now defined for the **c**, **s**, and **[** conversion specifiers.

31358 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the  
31359 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

31360 **Issue 6**

31361 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several  
31362 occurrences of "characters" in the text which have been replaced with the term "bytes".

31363 The normative text is updated to avoid use of the term "must" for application requirements.

31364 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 31365 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.
- 31366 • The DESCRIPTION is updated.
- 31367 • The **hh**, **ll**, **j**, **t**, and **z** length modifiers are added.
- 31368 • The **a**, **A**, and **F** conversion characters are added.

31369 The DESCRIPTION is updated to use the terms "conversion specifier" and "conversion  
31370 specification" consistently.

**fscanf()**31371 **Issue 7**

- 31372 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 31373 SD5-XSH-ERN-9 is applied, correcting *fscanf()* to *scanf()* in the DESCRIPTION.
- 31374 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.
- 31375 Functionality relating to the %n\$ form of conversion specification is moved from the XSI option  
31376 to the Base.
- 31377 Changes are made related to support for finegrained timestamps.
- 31378 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
31379 *malloc()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

31380 **NAME**

31381           fseek, fseeko — reposition a file-position indicator in a stream

31382 **SYNOPSIS**

31383           #include &lt;stdio.h&gt;

31384           int fseek(FILE \*stream, long offset, int whence);

31385 CX        int fseeko(FILE \*stream, off\_t offset, int whence);

31386 **DESCRIPTION**31387 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
31388 conflict between the requirements described here and the ISO C standard is unintentional. This  
31389 volume of POSIX.1-2008 defers to the ISO C standard.31390        The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a  
31391 read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.31392        The new position, measured in bytes from the beginning of the file, shall be obtained by adding  
31393 *offset* to the position specified by *whence*. The specified point is the beginning of the file for  
31394 SEEK\_SET, the current value of the file-position indicator for SEEK\_CUR, or end-of-file for  
31395 SEEK\_END.31396        If the stream is to be used with wide-character input/output functions, the application shall  
31397 ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and  
31398 *whence* is SEEK\_SET.31399        A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects  
31400 of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an  
31401 update stream may be either input or output.31402 CX        If the most recent operation, other than *ftell()*, on a given stream is *fflush()*, the file offset in the  
31403 underlying open file description shall be adjusted to reflect the location specified by *fseek()*.31404        The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing  
31405 data in the file. If data is later written at this point, subsequent reads of data in the gap shall  
31406 return bytes with the value 0 until data is actually written into the gap.31407        The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined.  
31408 The value of the file offset associated with such a device is undefined.31409        If the stream is writable and buffered data had not been written to the underlying file, *fseek()*  
31410 shall cause the unwritten data to be written to the file and shall mark the last data modification  
31411 and last file status change timestamps of the file for update.31412        In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is  
31413 implementation-defined.31414        The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is  
31415 of type **off\_t**.31416 **RETURN VALUE**31417 CX        The *fseek()* and *fseeko()* functions shall return 0 if they succeed.31418 CX        Otherwise, they shall return -1 and set *errno* to indicate the error.31419 **ERRORS**31420 CX        The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream's*  
31421 buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or  
31422 *write()* to be invoked, and:

**fseek()**

System Interfaces

|       |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
|-------|-----|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 31423 | CX  | [EAGAIN]    | The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.                                                                                                                                                                                                                                                      |
| 31424 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31425 | CX  | [EBADF]     | The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.                                                                                                                                                                                                                    |
| 31426 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31427 | CX  | [EFBIG]     | An attempt was made to write a file that exceeds the maximum file size.                                                                                                                                                                                                                                                                                         |
| 31428 | XSI | [EFBIG]     | An attempt was made to write a file that exceeds the file size limit of the process.                                                                                                                                                                                                                                                                            |
| 31429 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31430 | CX  | [EFBIG]     | The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.                                                                                                                                                                                                                           |
| 31431 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31432 | CX  | [EINTR]     | The write operation was terminated due to the receipt of a signal, and no data was transferred.                                                                                                                                                                                                                                                                 |
| 31433 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31434 | CX  | [EINVAL]    | The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.                                                                                                                                                                                                                                                  |
| 31435 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31436 | CX  | [EIO]       | A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions. |
| 31437 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31438 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31439 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31440 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31441 | CX  | [ENOSPC]    | There was no free space remaining on the device containing the file.                                                                                                                                                                                                                                                                                            |
| 31442 | CX  | [ENXIO]     | A request was made of a nonexistent device, or the request was outside the capabilities of the device.                                                                                                                                                                                                                                                          |
| 31443 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31444 | CX  | [EOVERFLOW] | For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>long</b> .                                                                                                                                                                                                                        |
| 31445 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31446 | CX  | [EOVERFLOW] | For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>off_t</b> .                                                                                                                                                                                                                      |
| 31447 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31448 | CX  | [EPIPE]     | An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.                                                                                                                                                                                                                  |
| 31449 |     |             |                                                                                                                                                                                                                                                                                                                                                                 |
| 31450 | CX  | [ESPIPE]    | The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.                                                                                                                                                                                                                                                                                 |

31451 **EXAMPLES**

31452 None.

31453 **APPLICATION USAGE**

31454 None.

31455 **RATIONALE**

31456 None.

31457 **FUTURE DIRECTIONS**

31458 None.

31459 **SEE ALSO**31460 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*

31461 XBD &lt;stdio.h&gt;

31462 **CHANGE HISTORY**

31463 First released in Issue 1. Derived from Issue 1 of the SVID.

31464 **Issue 5**

31465 Normative text previously in the APPLICATION USAGE section is moved to the  
31466 DESCRIPTION.

31467 Large File Summit extensions are added.

31468 **Issue 6**

31469 Extensions beyond the ISO C standard are marked.

31470 The following new requirements on POSIX implementations derive from alignment with the  
31471 Single UNIX Specification:

- 31472 • The *fseeko()* function is added.
- 31473 • The [EFBIG], [Eoverflow], and [ENXIO] mandatory error conditions are added.

31474 The following change is incorporated for alignment with the FIPS requirements:

- 31475 • The [EINTR] error is no longer an indication that the implementation does not report  
31476 partial transfers.

31477 The normative text is updated to avoid use of the term “must” for application requirements.

31478 The DESCRIPTION is updated to explicitly state that *seek()* sets the file-position indicator, and  
31479 then on error the error indicator is set and *fseek()* fails. This is for alignment with the  
31480 ISO/IEC 9899:1999 standard.

31481 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN]  
31482 error in the ERRORS section from “the process would be delayed” to “the thread would be  
31483 delayed”.

31484 **Issue 7**

31485 Changes are made related to support for finegrained timestamps.

**fsetpos()**31486 **NAME**31487 `fsetpos` — set current file position31488 **SYNOPSIS**31489 `#include <stdio.h>`31490 `int fsetpos(FILE *stream, const fpos_t *pos);`31491 **DESCRIPTION**31492 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31493 conflict between the requirements described here and the ISO C standard is unintentional. This  
31494 volume of POSIX.1-2008 defers to the ISO C standard.31495 The `fsetpos()` function shall set the file position and state indicators for the stream pointed to by  
31496 `stream` according to the value of the object pointed to by `pos`, which the application shall ensure is  
31497 a value obtained from an earlier call to `fgetpos()` on the same stream. If a read or write error  
31498 occurs, the error indicator for the stream shall be set and `fsetpos()` fails.31499 A successful call to the `fsetpos()` function shall clear the end-of-file indicator for the stream and  
31500 undo any effects of `ungetc()` on the same stream. After an `fsetpos()` call, the next operation on an  
31501 update stream may be either input or output.31502 CX The behavior of `fsetpos()` on devices which are incapable of seeking is implementation-defined.  
31503 The value of the file offset associated with such a device is undefined.31504 **RETURN VALUE**31505 The `fsetpos()` function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and  
31506 set `errno` to indicate the error.31507 **ERRORS**31508 CX The `fsetpos()` function shall fail if, either the `stream` is unbuffered or the `stream`'s buffer needed to  
31509 be flushed, and the call to `fsetpos()` causes an underlying `lseek()` or `write()` to be invoked, and:31510 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the thread would be  
31511 delayed in the write operation.31512 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the  
31513 stream's buffer needed to be flushed and the file is not open.

31514 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

31515 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the  
31516 process.31517 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
31518 offset maximum associated with the corresponding stream.31519 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
31520 was transferred.31521 CX [EIO] A physical I/O error has occurred, or the process is a member of a background  
31522 process group attempting to perform a `write()` to its controlling terminal,  
31523 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
31524 process group of the process is orphaned. This error may also be returned  
31525 under implementation-defined conditions.

31526 CX [ENOSPC] There was no free space remaining on the device containing the file.

31527 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
31528 capabilities of the device.

31529 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading  
31530 by any process; a SIGPIPE signal shall also be sent to the thread.

31531 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

#### 31532 EXAMPLES

31533 None.

#### 31534 APPLICATION USAGE

31535 None.

#### 31536 RATIONALE

31537 None.

#### 31538 FUTURE DIRECTIONS

31539 None.

#### 31540 SEE ALSO

31541 *fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write()*

31542 XBD <stdio.h>

#### 31543 CHANGE HISTORY

31544 First released in Issue 4. Derived from the ISO C standard.

#### 31545 Issue 6

31546 Extensions beyond the ISO C standard are marked.

31547 An additional [ESPIPE] error condition is added for sockets.

31548 The normative text is updated to avoid use of the term “must” for application requirements.

31549 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or  
31550 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

31551 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous  
31552 [EINVAL] error case from the ERRORS section.

31553 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN]  
31554 error in the ERRORS section from “the process would be delayed” to “the thread would be  
31555 delayed”.

#### 31556 Issue 7

31557 SD5-XSH-ERN-220 is applied.

**fstat()**31558 **NAME**31559 `fstat` — get file status31560 **SYNOPSIS**31561 `#include <sys/stat.h>`31562 `int fstat(int fildev, struct stat *buf);`31563 **DESCRIPTION**31564 The `fstat()` function shall obtain information about an open file associated with the file  
31565 descriptor *fildev*, and shall write it to the area pointed to by *buf*.31566 SHM If *fildev* references a shared memory object, the implementation shall update in the `stat` structure  
31567 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
31568 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
31569 valid. The implementation may update other fields and flags.31570 TYM If *fildev* references a typed memory object, the implementation shall update in the `stat` structure  
31571 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
31572 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
31573 valid. The implementation may update other fields and flags.31574 The *buf* argument is a pointer to a `stat` structure, as defined in `<sys/stat.h>`, into which  
31575 information is placed concerning the file.31576 For all other file types defined in this volume of POSIX.1-2008, the structure members *st\_mode*,  
31577 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
31578 value of the *st\_nlink* member shall be set to the number of links to the file.31579 An implementation that provides additional or alternative file access control mechanisms may,  
31580 under implementation-defined conditions, cause `fstat()` to fail.31581 The `fstat()` function shall update any time-related fields (as described in XBD Section 4.8, on  
31582 page 109), before writing into the `stat` structure.31583 **RETURN VALUE**31584 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
31585 indicate the error.31586 **ERRORS**31587 The `fstat()` function shall fail if:31588 [EBADF] The *fildev* argument is not a valid file descriptor.

31589 [EIO] An I/O error occurred while reading from the file system.

31590 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
31591 serial number cannot be represented correctly in the structure pointed to by  
31592 *buf*.31593 The `fstat()` function may fail if:31594 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*  
31595 argument.

31596 **EXAMPLES**31597 **Obtaining File Status Information**

31598 The following example shows how to obtain file status information for a file named  
 31599 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The  
 31600 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file  
 31601 descriptor *fildes*.

```
31602 #include <sys/types.h>
31603 #include <sys/stat.h>
31604 #include <fcntl.h>

31605 struct stat buffer;
31606 int      status;
31607 ...
31608 fildes = open("/home/cnd/mod1", O_RDWR);
31609 status = fstat(fildes, &buffer);
```

31610 **APPLICATION USAGE**

31611 None.

31612 **RATIONALE**

31613 None.

31614 **FUTURE DIRECTIONS**

31615 None.

31616 **SEE ALSO**

31617 [fstatat\(\)](#)

31618 XBD [Section 4.8](#) (on page 109), [<sys/stat.h>](#), [<sys/types.h>](#)

31619 **CHANGE HISTORY**

31620 First released in Issue 1. Derived from Issue 1 of the SVID.

31621 **Issue 5**

31622 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

31623 Large File Summit extensions are added.

31624 **Issue 6**

31625 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

31626 The following new requirements on POSIX implementations derive from alignment with the  
 31627 Single UNIX Specification:

- 31628 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 31629 required for conforming implementations of previous POSIX specifications, it was not  
 31630 required for UNIX applications.
- 31631 • The [EIO] mandatory error condition is added.
- 31632 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 31633 files.
- 31634 • The [EOVERFLOW] optional error condition is added.

31635 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 31636 shared memory object semantics apply to typed memory objects.

**fstat()**31637 **Issue 7**

31638 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st\_nlink*  
31639 applies.

31640 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 31641 NAME

31642 fstatat, lstat, stat — get file status

## 31643 SYNOPSIS

```
31644 #include <sys/stat.h>
31645 int fstatat(int fd, const char *restrict path,
31646             struct stat *restrict buf, int flag);
31647 int lstat(const char *restrict path, struct stat *restrict buf);
31648 int stat(const char *restrict path, struct stat *restrict buf);
```

## 31649 DESCRIPTION

31650 The *stat()* function shall obtain information about the named file and write it to the area pointed  
 31651 to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or  
 31652 execute permission of the named file is not required. An implementation that provides  
 31653 additional or alternate file access control mechanisms may, under implementation-defined  
 31654 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 31655 specified by *path*.

31656 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using  
 31657 the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 31658 the file exists.

31659 The *buf* argument is a pointer to a **stat** structure, as defined in the `<sys/stat.h>` header, into  
 31660 which information is placed concerning the file.

31661 The *stat()* function shall update any time-related fields (as described in XBD Section 4.8, on page  
 31662 109), before writing into the **stat** structure.

31663 SHM If the named file is a shared memory object, the implementation shall update in the **stat** structure  
 31664 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 31665 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 31666 valid. The implementation may update other fields and flags.

31667 TYM If the named file is a typed memory object, the implementation shall update in the **stat** structure  
 31668 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 31669 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 31670 valid. The implementation may update other fields and flags.

31671 For all other file types defined in this volume of POSIX.1-2008, the structure members *st\_mode*,  
 31672 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
 31673 value of the member *st\_nlink* shall be set to the number of links to the file.

31674 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In  
 31675 that case *lstat()* shall return information about the link, while *stat()* shall return information  
 31676 about the file the link references.

31677 For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 31678 the file type macros. The file mode bits in *st\_mode* are unspecified. The structure members *st\_ino*,  
 31679 *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the value of  
 31680 the *st\_nlink* member shall be set to the number of (hard) links to the symbolic link. The value of  
 31681 the *st\_size* member shall be set to the length of the pathname contained in the symbolic link not  
 31682 including any terminating null byte.

31683 The *fstatat()* function shall be equivalent to the *stat()* or *lstat()* function, depending on the value  
 31684 of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status  
 31685 shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead  
 31686 of the current working directory. If the file descriptor was opened without O\_SEARCH, the

**fstatat()**

31687 function shall check whether directory searches are permitted using the current permissions of  
 31688 the directory underlying the file descriptor. If the file descriptor was opened with O\_SEARCH,  
 31689 the function shall not perform the check.

31690 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 31691 in **<fcntl.h>**:

31692 AT\_SYMLINK\_NOFOLLOW

31693 If *path* names a symbolic link, the status of the symbolic link is returned.

31694 If *fstatat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 31695 directory is used and the behavior shall be identical to a call to *stat()* or *lstat()* respectively,  
 31696 depending on whether or not the AT\_SYMLINK\_NOFOLLOW bit is set in *flag*.

**RETURN VALUE**

31697 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 31698 return -1 and set *errno* to indicate the error.

**ERRORS**

31700 These functions shall fail if:

31701 [EACCES] Search permission is denied for a component of the path prefix.

31702 [EIO] An error occurred while reading from the file system.

31703 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 31704 argument.

31705 [ENAMETOOLONG]

31706 The length of a component of a pathname is longer than {NAME\_MAX}.

31707 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

31708 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
 31709 contains at least one non-*<slash>* character and ends with one or more trailing  
 31710 *<slash>* characters and the last pathname component names an existing file  
 31711 that is neither a directory nor a symbolic link to a directory.

31712 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
 31713 serial number cannot be represented correctly in the structure pointed to by  
 31714 *buf*.

31715 The *fstatat()* function shall fail if:

31716 [EACCES] *fd* was not opened with O\_SEARCH and the permissions of the directory  
 31717 underlying *fd* do not permit directory searches.

31718 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
 31719 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.

31720 These functions may fail if:

31721 [ELOOP] More than {SYMLINK\_MAX} symbolic links were encountered during  
 31722 resolution of the *path* argument.

31723 [ENAMETOOLONG]

31724 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 31725 symbolic link produced an intermediate result with a length that exceeds  
 31726 {PATH\_MAX}.

- 31728 [Eoverflow] A value to be stored would overflow one of the members of the **stat** structure.
- 31729 The *fstatat()* function may fail if:
- 31730 [EINVAL] The value of the *flag* argument is not valid.
- 31731 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
31732 file descriptor associated with a directory.

31733 **EXAMPLES**31734 **Obtaining File Status Information**

31735 The following example shows how to obtain file status information for a file named  
31736 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
31737 #include <sys/types.h>
31738 #include <sys/stat.h>
31739 #include <fcntl.h>

31740 struct stat buffer;
31741 int      status;
31742 ...
31743 status = stat("/home/cnd/mod1", &buffer);
```

31744 **Getting Directory Information**

31745 The following example fragment gets status information for each entry in a directory. The call to  
31746 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
31747 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the  
31748 program.

```
31749 #include <sys/types.h>
31750 #include <sys/stat.h>
31751 #include <dirent.h>
31752 #include <pwd.h>
31753 #include <grp.h>
31754 #include <time.h>
31755 #include <locale.h>
31756 #include <langinfo.h>
31757 #include <stdio.h>
31758 #include <stdint.h>

31759 struct dirent *dp;
31760 struct stat   statbuf;
31761 struct passwd *pwd;
31762 struct group  *grp;
31763 struct tm     *tm;
31764 char          datestring[256];
31765 ...
31766 /* Loop through directory entries. */
31767 while ((dp = readdir(dir)) != NULL) {

31768     /* Get entry's information. */
31769     if (stat(dp->d_name, &statbuf) == -1)
31770         continue;
```

**fstatat()**

```

31771     /* Print out type, permissions, and number of links. */
31772     printf("%10.10s", sperm (statbuf.st_mode));
31773     printf("%4d", statbuf.st_nlink);

31774     /* Print out owner's name if it is found using getpwuid(). */
31775     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
31776         printf(" %-8.8s", pwd->pw_name);
31777     else
31778         printf(" %-8d", statbuf.st_uid);

31779     /* Print out group name if it is found using getgrgid(). */
31780     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
31781         printf(" %-8.8s", grp->gr_name);
31782     else
31783         printf(" %-8d", statbuf.st_gid);

31784     /* Print size of file. */
31785     printf(" %9jd", (intmax_t)statbuf.st_size);

31786     tm = localtime(&statbuf.st_mtime);

31787     /* Get localized date string. */
31788     strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);

31789     printf(" %s %s\n", datestring, dp->d_name);
31790 }

```

**Obtaining Symbolic Link Status Information**

The following example shows how to obtain status information for a symbolic link named **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path* argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**), the results of calling the function would be the same as those returned by a call to the *stat()* function.

```

31797 #include <sys/stat.h>
31798 struct stat buffer;
31799 int status;
31800 ...
31801 status = lstat("/modules/pass1", &buffer);

```

**APPLICATION USAGE**

None.

**RATIONALE**

The intent of the paragraph describing “additional or alternate file access control mechanisms” is to allow a secure implementation where a process with a label that does not dominate the file’s label cannot perform a *stat()* function. This is not related to read permission; a process with a label that dominates the file’s label does not need read permission. An implementation that supports write-up operations could fail *fstat()* function calls even though it has a valid file descriptor open for writing.

The *lstat()* function is not required to update the time-related fields if the named file is not a symbolic link. While the *st\_uid*, *st\_gid*, *st\_atim*, *st\_mtim*, and *st\_ctim* members of the **stat** structure may apply to a symbolic link, they are not required to do so. No functions in POSIX.1-2008 are required to maintain any of these time fields.

31815 The purpose of the *fstatat()* function is to obtain the status of files in directories other than the  
 31816 current working directory without exposure to race conditions. Any part of the path of a file  
 31817 could be changed in parallel to a call to *stat()*, resulting in unspecified behavior. By opening a  
 31818 file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that  
 31819 the file for which status is returned is located relative to the desired directory.

#### 31820 FUTURE DIRECTIONS

31821 None.

#### 31822 SEE ALSO

31823 *access()*, *chmod()*, *fdopendir()*, *fstat()*, *mknod()*, *readlink()*, *symlink()*

31824 XBD Section 4.8 (on page 109), `<fcntl.h>`, `<sys/stat.h>`, `<sys/types.h>`

#### 31825 CHANGE HISTORY

31826 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 31827 Issue 5

31828 Large File Summit extensions are added.

#### 31829 Issue 6

31830 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

31831 The following new requirements on POSIX implementations derive from alignment with the  
 31832 Single UNIX Specification:

- 31833 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 31834 required for conforming implementations of previous POSIX specifications, it was not  
 31835 required for UNIX applications.
- 31836 • The [EIO] mandatory error condition is added.
- 31837 • The [ELOOP] mandatory error condition is added.
- 31838 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 31839 files.
- 31840 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are  
 31841 added.

31842 The following changes were made to align with the IEEE P1003.1a draft standard:

- 31843 • Details are added regarding the treatment of symbolic links.
- 31844 • The [ELOOP] optional error condition is added.

31845 The normative text is updated to avoid use of the term “must” for application requirements.

31846 The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999  
 31847 standard.

#### 31848 Issue 7

31849 Austin Group Interpretation 1003.1-2001 #143 is applied.

31850 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st\_nlink*  
 31851 applies.

31852 The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 31853 Set Part 2.

31854 Changes are made related to support for finegrained timestamps.

31855 The *lstat()* function is now required to return meaningful data for symbolic links in all **stat**

**fstatat()**

- 31856 structure fields, except for the permission bits of *st\_mode*.
- 31857 Changes are made to allow a directory to be opened for searching.
- 31858 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
31859 pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

31860 **NAME**31861 `fstatvfs, statvfs` — get file system information31862 **SYNOPSIS**31863 `#include <sys/statvfs.h>`31864 `int fstatvfs(int fildev, struct statvfs *buf);`31865 `int statvfs(const char *restrict path, struct statvfs *restrict buf);`31866 **DESCRIPTION**31867 The `fstatvfs()` function shall obtain information about the file system containing the file  
31868 referenced by *fildev*.31869 The `statvfs()` function shall obtain information about the file system containing the file named by  
31870 *path*.31871 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,  
31872 write, or execute permission of the named file is not required.31873 The following flags can be returned in the *f\_flag* member:31874 **ST\_RDONLY** Read-only file system.31875 **ST\_NOSUID** Setuid/setgid bits ignored by *exec*.31876 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file  
31877 systems.31878 **RETURN VALUE**31879 Upon successful completion, `statvfs()` shall return 0. Otherwise, it shall return -1 and set *errno* to  
31880 indicate the error.31881 **ERRORS**31882 The `fstatvfs()` and `statvfs()` functions shall fail if:31883 **[EIO]** An I/O error occurred while reading the file system.31884 **[EINTR]** A signal was caught during execution of the function.31885 **[EOVERFLOW]** One of the values to be returned cannot be represented correctly in the  
31886 structure pointed to by *buf*.31887 The `fstatvfs()` function shall fail if:31888 **[EBADF]** The *fildev* argument is not an open file descriptor.31889 The `statvfs()` function shall fail if:31890 **[EACCES]** Search permission is denied on a component of the path prefix.31891 **[ELOOP]** A loop exists in symbolic links encountered during resolution of the *path*  
31892 argument.31893 **[ENAMETOOLONG]**

31894 The length of a component of a pathname is longer than {NAME\_MAX}.

31895 **[ENOENT]** A component of *path* does not name an existing file or *path* is an empty string.31896 **[ENOTDIR]** A component of the path prefix is not a directory, or the *path* argument  
31897 contains at least one non-`<slash>` character and ends with one or more trailing  
31898 `<slash>` characters and the last pathname component names an existing file  
31899 that is neither a directory nor a symbolic link to a directory.

**fstatvfs()**

- 31900 The *statvfs()* function may fail if:
- 31901 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
31902 resolution of the *path* argument.
- 31903 [ENAMETOOLONG]  
31904 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
31905 symbolic link produced an intermediate result with a length that exceeds  
31906 {PATH\_MAX}.

31907 **EXAMPLES**31908 **Obtaining File System Information Using fstatvfs()**

31909 The following example shows how to obtain file system information for the file system upon  
31910 which the file named **/home/cnd/mod1** resides, using the *fstatvfs()* function. The  
31911 **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed  
31912 to the *fstatvfs()* function.

```
31913 #include <sys/statvfs.h>
31914 #include <fcntl.h>
31915
31916 struct statvfs buffer;
31917 int status;
31918 ...
31919 fildes = open("/home/cnd/mod1", O_RDWR);
31920 status = fstatvfs(fildes, &buffer);
```

31920 **Obtaining File System Information Using statvfs()**

31921 The following example shows how to obtain file system information for the file system upon  
31922 which the file named **/home/cnd/mod1** resides, using the *statvfs()* function.

```
31923 #include <sys/statvfs.h>
31924
31925 struct statvfs buffer;
31926 int status;
31927 ...
31928 status = statvfs("/home/cnd/mod1", &buffer);
```

31928 **APPLICATION USAGE**

31929 None.

31930 **RATIONALE**

31931 None.

31932 **FUTURE DIRECTIONS**

31933 None.

31934 **SEE ALSO**

31935 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,  
31936 *unlink()*, *utime()*, *write()*

31937 XBD [<sys/statvfs.h>](#)

31938 **CHANGE HISTORY**

31939 First released in Issue 4, Version 2.

31940 **Issue 5**

31941 Moved from X/OPEN UNIX extension to BASE.

31942 Large File Summit extensions are added.

31943 **Issue 6**

31944 The normative text is updated to avoid use of the term “must” for application requirements.

31945 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the  
31946 ISO/IEC 9899: 1999 standard.

31947 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
31948 [ELOOP] error condition is added.

31949 **Issue 7**

31950 Austin Group Interpretation 1003.1-2001 #143 is applied.

31951 SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

31952 The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.

31953 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
31954 pathname exists but is not a directory or a symbolic link to a directory.

**fsync()**31955 **NAME**

31956 fsync — synchronize changes to a file

31957 **SYNOPSIS**

```
31958 FSC #include <unistd.h>
31959 int fsync(int fildes);
```

31960 **DESCRIPTION**

31961 The *fsync()* function shall request that all data for the open file descriptor named by *fildes* is to be  
 31962 transferred to the storage device associated with the file described by *fildes*. The nature of the  
 31963 transfer is implementation-defined. The *fsync()* function shall not return until the system has  
 31964 completed that action or until an error is detected.

31965 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued  
 31966 I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O  
 31967 completion state. All I/O operations shall be completed as defined for synchronized I/O file  
 31968 integrity completion.

31969 **RETURN VALUE**

31970 Upon successful completion, *fsync()* shall return 0. Otherwise, -1 shall be returned and *errno* set  
 31971 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed  
 31972 to have been completed.

31973 **ERRORS**31974 The *fsync()* function shall fail if:

- 31975 [EBADF] The *fildes* argument is not a valid descriptor.
- 31976 [EINTR] The *fsync()* function was interrupted by a signal.
- 31977 [EINVAL] The *fildes* argument does not refer to a file on which this operation is possible.
- 31978 [EIO] An I/O error occurred while reading from or writing to the file system.

31979 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions  
 31980 defined for *read()* and *write()*.

31981 **EXAMPLES**

31982 None.

31983 **APPLICATION USAGE**

31984 The *fsync()* function should be used by programs which require modifications to a file to be  
 31985 completed before continuing; for example, a program which contains a simple transaction  
 31986 facility might use it to ensure that all modifications to a file or files caused by a transaction are  
 31987 recorded.

31988 **RATIONALE**

31989 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to  
 31990 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is  
 31991 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and  
 31992 “non-volatile storage” are not defined here, the wording has to be more abstract.

31993 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance  
 31994 document to tell the user what can be expected from the system. It is explicitly intended that a  
 31995 null implementation is permitted. This could be valid in the case where the system cannot assure  
 31996 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the  
 31997 functionality is not required. In the middle ground between these extremes, *fsync()* might or  
 31998 might not actually cause data to be written where it is safe from a power failure. The

31999 conformance document should identify at least that one configuration exists (and how to obtain  
 32000 that configuration) where this can be assured for at least some files that the user can select to use  
 32001 for critical data. It is not intended that an exhaustive list is required, but rather sufficient  
 32002 information is provided so that if critical data needs to be saved, the user can determine how the  
 32003 system is to be configured to allow the data to be written to non-volatile storage.

32004 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.  
 32005 That does not make the function any less valuable, just more difficult to test. A formal  
 32006 conformance test should probably force a system crash (power shutdown) during the test for  
 32007 this condition, but it needs to be done in such a way that automated testing does not require this  
 32008 to be done except when a formal record of the results is being made. It would also not be  
 32009 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation  
 32010 issue.

#### 32011 FUTURE DIRECTIONS

32012 None.

#### 32013 SEE ALSO

32014 [sync\(\)](#)

32015 XBD [<unistd.h>](#)

#### 32016 CHANGE HISTORY

32017 First released in Issue 3.

#### 32018 Issue 5

32019 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and  
 32020 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate  
 32021 that *fsync()* can return the error conditions defined for *read()* and *write()*.

#### 32022 Issue 6

32023 This function is marked as part of the File Synchronization option.

32024 The following new requirements on POSIX implementations derive from alignment with the  
 32025 Single UNIX Specification:

- 32026 • The [EINVAL] and [EIO] mandatory error conditions are added.

32027 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial  
 32028 rewording of the DESCRIPTION. No change in meaning is intended.

**ftell()**32029 **NAME**32030 `ftell, ftello` — return a file offset in a stream32031 **SYNOPSIS**32032 `#include <stdio.h>`32033 `long ftell(FILE *stream);`32034 CX `off_t ftello(FILE *stream);`32035 **DESCRIPTION**32036 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
32037 conflict between the requirements described here and the ISO C standard is unintentional. This  
32038 volume of POSIX.1-2008 defers to the ISO C standard.32039 The `ftell()` function shall obtain the current value of the file-position indicator for the stream  
32040 pointed to by `stream`.32041 CX The `ftello()` function shall be equivalent to `ftell()`, except that the return value is of type `off_t`.32042 **RETURN VALUE**32043 CX Upon successful completion, `ftell()` and `ftello()` shall return the current value of the file-position  
32044 indicator for the stream measured in bytes from the beginning of the file.32045 CX Otherwise, `ftell()` and `ftello()` shall return `-1`, cast to `long` and `off_t` respectively, and set `errno` to  
32046 indicate the error.32047 **ERRORS**32048 CX The `ftell()` and `ftello()` functions shall fail if:32049 CX [EBADF] The file descriptor underlying `stream` is not an open file descriptor.32050 CX [EOVERFLOW] For `ftell()`, the current file offset cannot be represented correctly in an object of  
32051 type `long`.32052 CX [EOVERFLOW] For `ftello()`, the current file offset cannot be represented correctly in an object  
32053 of type `off_t`.32054 CX [ESPIPE] The file descriptor underlying `stream` is associated with a pipe or FIFO.32055 The `ftell()` function may fail if:32056 CX [ESPIPE] The file descriptor underlying `stream` is associated with a socket.32057 **EXAMPLES**

32058 None.

32059 **APPLICATION USAGE**

32060 None.

32061 **RATIONALE**

32062 None.

32063 **FUTURE DIRECTIONS**

32064 None.

32065 **SEE ALSO**32066 `fgetpos()`, `fopen()`, `fseek()`, `lseek()`32067 XBD `<stdio.h>`

32068 **CHANGE HISTORY**

32069 First released in Issue 1. Derived from Issue 1 of the SVID.

32070 **Issue 5**

32071 Large File Summit extensions are added.

32072 **Issue 6**

32073 Extensions beyond the ISO C standard are marked.

32074 The following new requirements on POSIX implementations derive from alignment with the  
32075 Single UNIX Specification:

- 32076 • The *ftello()* function is added.
- 32077 • The [Eoverflow] error conditions are added.

32078 An additional [ESPIPE] error condition is added for sockets.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ftok()**32079 **NAME**32080 `ftok` — generate an IPC key32081 **SYNOPSIS**

```
32082 xSI #include <sys/ipc.h>
32083 key_t ftok(const char *path, int id);
```

32084 **DESCRIPTION**

32085 The `ftok()` function shall return a key based on *path* and *id* that is usable in subsequent calls to  
 32086 `msgget()`, `semget()`, and `shmget()`. The application shall ensure that the *path* argument is the  
 32087 pathname of an existing file that the process is able to `stat()`.

32088 The `ftok()` function shall return the same key value for all paths that name the same file, when  
 32089 called with the same *id* value, and return different key values when called with different *id*  
 32090 values or with paths that name different files existing on the same file system at the same time. It  
 32091 is unspecified whether `ftok()` shall return the same key value when called again after the file  
 32092 named by *path* is removed and recreated with the same name.

32093 Only the low-order 8-bits of *id* are significant. The behavior of `ftok()` is unspecified if these bits  
 32094 are 0.

32095 **RETURN VALUE**

32096 Upon successful completion, `ftok()` shall return a key. Otherwise, `ftok()` shall return `(key_t)-1`  
 32097 and set `errno` to indicate the error.

32098 **ERRORS**

32099 The `ftok()` function shall fail if:

- 32100 [EACCES] Search permission is denied for a component of the path prefix.
- 32101 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 32102 argument.
- 32103 [ENAMETOOLONG]  
 32104 The length of a component of a pathname is longer than {NAME\_MAX}.
- 32105 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 32106 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
 32107 contains at least one non-`<slash>` character and ends with one or more trailing  
 32108 `<slash>` characters and the last pathname component names an existing file  
 32109 that is neither a directory nor a symbolic link to a directory.

32110 The `ftok()` function may fail if:

- 32111 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 32112 resolution of the *path* argument.
- 32113 [ENAMETOOLONG]  
 32114 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 32115 symbolic link produced an intermediate result with a length that exceeds  
 32116 {PATH\_MAX}.

32117 **EXAMPLES**32118 **Getting an IPC Key**

32119 The following example gets a unique key that can be used by the IPC functions *semget()*,  
 32120 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*  
 32121 and the pathname */tmp*.

```
32122 #include <sys/ipc.h>
32123 ...
32124 key_t key;
32125 char *path = "/tmp";
32126 int id = 'S';
32127 key = ftok(path, id);
```

32128 **Saving an IPC Key**

32129 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It  
 32130 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a  
 32131 later call to *semget()*, *msgget()*, or *shmget()*.

```
32132 #include <sys/ipc.h>
32133 ...
32134 key_t semkey;
32135 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
32136     perror("IPC error: ftok"); exit(1);
32137 }
```

32138 **APPLICATION USAGE**

32139 For maximum portability, *id* should be a single-byte character.

32140 **RATIONALE**

32141 None.

32142 **FUTURE DIRECTIONS**

32143 None.

32144 **SEE ALSO**

32145 *msgget()*, *semget()*, *shmget()*

32146 XBD *<sys/ipc.h>*

32147 **CHANGE HISTORY**

32148 First released in Issue 4, Version 2.

32149 **Issue 5**

32150 Moved from X/OPEN UNIX extension to BASE.

32151 **Issue 6**

32152 The normative text is updated to avoid use of the term “must” for application requirements.

32153 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 32154 [ELOOP] error condition is added.

**ftok()**

32155 **Issue 7**

32156 Austin Group Interpretation 1003.1-2001 #143 is applied.

32157 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
32158 pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

32159 **NAME**

32160 ftruncate — truncate a file to a specified length

32161 **SYNOPSIS**

32162 #include &lt;unistd.h&gt;

32163 int ftruncate(int *filde*s, off\_t *length*);32164 **DESCRIPTION**32165 If *filde*s is not a valid file descriptor open for writing, the *ftruncate*(*)* function shall fail.

32166 If *filde*s refers to a regular file, the *ftruncate*(*)* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate*(*)* shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate*(*)*.

32172 Upon successful completion, if *filde*s refers to a regular file, *ftruncate*(*)* shall mark for update the last data modification and last file status change timestamps of the file and the S\_ISUID and S\_ISGID bits of the file mode may be cleared. If the *ftruncate*(*)* function is unsuccessful, the file is unaffected.

32176 XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

32178 If *filde*s refers to a directory, *ftruncate*(*)* shall fail.32179 If *filde*s refers to any other file type, except a shared memory object, the result is unspecified.

32180 SHM If *filde*s refers to a shared memory object, *ftruncate*(*)* shall set the size of the shared memory object to *length*.

32182 SHM If the effect of *ftruncate*(*)* is to decrease the size of a memory mapped file or a shared memory object and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

32185 References to discarded pages shall result in the generation of a SIGBUS signal.

32186 If the effect of *ftruncate*(*)* is to increase the size of a memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

32189 **RETURN VALUE**

32190 Upon successful completion, *ftruncate*(*)* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

32192 **ERRORS**32193 The *ftruncate*(*)* function shall fail if:

32194 [EINTR] A signal was caught during execution.

32195 [EINVAL] The *length* argument was less than 0.

32196 [EFBIG] or [EINVAL]

32197 The *length* argument was greater than the maximum file size.

32198 [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *filde*s.

32199

**ftruncate()**

32200 [EIO] An I/O error occurred while reading from or writing to a file system.

32201 [EBADF] or [EINVAL]

32202 The *files* argument is not a file descriptor open for writing.

32203 **EXAMPLES**

32204 None.

32205 **APPLICATION USAGE**

32206 None.

32207 **RATIONALE**

32208 None.

32209 **FUTURE DIRECTIONS**

32210 None.

32211 **SEE ALSO**

32212 *open()*, *truncate()*

32213 XBD <*unistd.h*>

32214 **CHANGE HISTORY**

32215 First released in Issue 4, Version 2.

32216 **Issue 5**

32217 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX  
32218 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is  
32219 added to the list of mandatory errors that can be returned by *ftruncate()*.

32220 Large File Summit extensions are added.

32221 **Issue 6**

32222 The *truncate()* function is split out into a separate reference page.

32223 The following new requirements on POSIX implementations derive from alignment with the  
32224 Single UNIX Specification:

- 32225 • The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a  
32226 regular file, the S\_ISUID and S\_ISGID bits in the file mode may be cleared.

32227 The following changes were made to align with the IEEE P1003.1a draft standard:

- 32228 • The DESCRIPTION text is updated.

32229 XSI-conformant systems are required to increase the size of the file if the file was previously  
32230 smaller than the size requested.

32231 **Issue 7**

32232 Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although  
32233 the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).

32234 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to  
32235 the Base.

32236 The DESCRIPTION is updated so that a call to *ftruncate()* when the file is smaller than the size  
32237 requested will increase the size of the file. Previously, non-XSI-conforming implementations  
32238 were allowed to increase the size of the file or fail.

32239 Changes are made related to support for finegrained timestamps.

32240 **NAME**

32241 ftrylockfile — stdio locking functions

32242 **SYNOPSIS**

```
32243 CX #include <stdio.h>  
32244 int ftrylockfile(FILE *file);
```

32245 **DESCRIPTION**32246 Refer to *flockfile()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ftw()**32247 **NAME**32248 **ftw** — traverse (walk) a file tree32249 **SYNOPSIS**

```
32250 OB XSI #include <ftw.h>
32251 int ftw(const char *path, int (*fn)(const char *,
32252     const struct stat *ptr, int flag), int ndirs);
```

32253 **DESCRIPTION**

32254 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each  
 32255 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a null-  
 32256 terminated character string containing the name of the object, a pointer to a **stat** structure  
 32257 containing information about the object, filled in as if *stat()* or *lstat()* had been called to retrieve  
 32258 the information. Possible values of the integer, defined in the **<ftw.h>** header, are:

32259 **FTW\_D** For a directory.32260 **FTW\_DNR** For a directory that cannot be read.32261 **FTW\_F** For a file.32262 **FTW\_SL** For a symbolic link (but see also **FTW\_NS** below).

32263 **FTW\_NS** For an object other than a symbolic link on which *stat()* could not successfully be  
 32264 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether  
 32265 *ftw()* passes **FTW\_SL** or **FTW\_NS** to the user-supplied function.

32266 If the integer is **FTW\_DNR**, descendants of that directory shall not be processed. If the integer is  
 32267 **FTW\_NS**, the **stat** structure contains undefined values. An example of an object that would  
 32268 cause **FTW\_NS** to be passed to the function pointed to by *fn* would be a file in a directory with  
 32269 read but without execute (search) permission.

32270 The *ftw()* function shall visit a directory before visiting any of its descendants.32271 The *ftw()* function shall use at most one file descriptor for each level in the tree.32272 The argument *ndirs* should be in the range [1, {**OPEN\_MAX**}].

32273 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a  
 32274 non-zero value, or some error, other than [**EACCES**], is detected within *ftw()*.

32275 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors  
 32276 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any  
 32277 directory streams and file descriptors it uses not counting any opened by the application-  
 32278 supplied *fn* function.

32279 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 32280 working directory.

32281 The *ftw()* function need not be thread-safe.32282 **RETURN VALUE**

32283 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero  
 32284 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function  
 32285 pointed to by *fn*. If *ftw()* detects an error, it shall return **-1** and set *errno* to indicate the error.

32286 If *ftw()* encounters an error other than [**EACCES**] (see **FTW\_DNR** and **FTW\_NS** above), it shall  
 32287 return **-1** and set *errno* to indicate the error. The external variable *errno* may contain any error  
 32288 value that is possible when a directory is opened or when one of the *stat* functions is executed on

32289 a directory or file.

### 32290 ERRORS

32291 The *ftw()* function shall fail if:

32292 [EACCES] Search permission is denied for any component of *path* or read permission is  
32293 denied for *path*.

32294 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
32295 argument.

32296 [ENAMETOOLONG]

32297 The length of a component of a pathname is longer than {NAME\_MAX}.

32298 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

32299 [ENOTDIR] A component of *path* is not a directory.

32300 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
32301 programming environment for one or more files found in the file hierarchy.

32302 The *ftw()* function may fail if:

32303 [EINVAL] The value of the *ndirs* argument is invalid.

32304 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
32305 resolution of the *path* argument.

32306 [ENAMETOOLONG]

32307 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
32308 symbolic link produced an intermediate result with a length that exceeds  
32309 {PATH\_MAX}.

32310 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set  
32311 accordingly.

### 32312 EXAMPLES

#### 32313 Walking a Directory Structure

32314 The following example walks the current directory structure, calling the *fn* function for every  
32315 directory entry, using at most 10 file descriptors:

```
32316 #include <ftw.h>
32317 ...
32318 if (ftw(".", fn, 10) != 0) {
32319     perror("ftw"); exit(2);
32320 }
```

### 32321 APPLICATION USAGE

32322 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly  
32323 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
32324 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains  
32325 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
32326 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
32327 invocation.

32328 Applications should use the *nftw()* function instead of the obsolescent *ftw()* function.

**ftw()**32329 **RATIONALE**

32330 None.

32331 **FUTURE DIRECTIONS**32332 The *ftw()* function may be removed in a future version.32333 **SEE ALSO**32334 *fdopendir()*, *fstatat()*, *longjmp()*, *nftw()*, *siglongjmp()*32335 XBD <*ftw.h*>, <*sys/stat.h*>32336 **CHANGE HISTORY**

32337 First released in Issue 1. Derived from Issue 1 of the SVID.

32338 **Issue 5**

32339 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

32340 **Issue 6**

32341 The ERRORS section is updated as follows:

- 32342 • The wording of the mandatory [ELOOP] error condition is updated.
- 32343 • A second optional [ELOOP] error condition is added.
- 32344 • The [EOVERFLOW] mandatory error condition is added.

32345 A note is added to the DESCRIPTION indicating that this function need not be reentrant, and  
 32346 that the results are unspecified if the application-supplied *fn* function does not preserve the  
 32347 current working directory.

32348 **Issue 7**

32349 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

32350 SD5-XBD-ERN-61 is applied.

32351 The *ftw()* function is marked obsolescent.

32352 **NAME**

32353 funlockfile — stdio locking functions

32354 **SYNOPSIS**

```
32355 CX #include <stdio.h>  
32356 void funlockfile(FILE *file);
```

32357 **DESCRIPTION**32358 Refer to *flockfile()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**futimens()**32359 **NAME**

32360 futimens, utimensat, utimes — set file access and modification times

32361 **SYNOPSIS**

32362 #include &lt;sys/stat.h&gt;

32363 int futimens(int *fd*, const struct timespec *times*[2]);32364 int utimensat(int *fd*, const char \**path*, const struct timespec *times*[2],32365 int *flag*);

32366 XSI #include &lt;sys/time.h&gt;

32367 int utimes(const char \**path*, const struct timeval *times*[2]);32368 **DESCRIPTION**

32369 The *futimens()* and *utimensat()* functions shall set the access and modification times of a file to  
 32370 the values of the *times* argument. The *futimens()* function changes the times of the file associated  
 32371 with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by  
 32372 the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions  
 32373 allow time specifications accurate to the nanosecond.

32374 For *futimens()* and *utimensat()*, the *times* argument is an array of two **timespec** structures. The  
 32375 first array member represents the date and time of last access, and the second member  
 32376 represents the date and time of last modification. The times in the **timespec** structure are  
 32377 measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set  
 32378 to the greatest value supported by the file system that is not greater than the specified time.

32379 If the *tv\_nsec* field of a **timespec** structure has the special value **UTIME\_NOW**, the file's relevant  
 32380 timestamp shall be set to the greatest value supported by the file system that is not greater than  
 32381 the current time. If the *tv\_nsec* field has the special value **UTIME\_OMIT**, the file's relevant  
 32382 timestamp shall not be changed. In either case, the *tv\_sec* field shall be ignored.

32383 If the *times* argument is a null pointer, both the access and modification timestamps shall be set  
 32384 to the greatest value supported by the file system that is not greater than the current time. If  
 32385 *utimensat()* is passed a relative path in the *path* argument, the file to be used shall be relative to  
 32386 the directory associated with the file descriptor *fd* instead of the current working directory. If the  
 32387 file descriptor was opened without **O\_SEARCH**, the function shall check whether directory  
 32388 searches are permitted using the current permissions of the directory underlying the file  
 32389 descriptor. If the file descriptor was opened with **O\_SEARCH**, the function shall not perform the  
 32390 check.

32391 If *utimensat()* is passed the special value **AT\_FDCWD** in the *fd* parameter, the current working  
 32392 directory shall be used.

32393 Only a process with the effective user ID equal to the user ID of the file, or with write access to  
 32394 the file, or with appropriate privileges may use *futimens()* or *utimensat()* with a null pointer as  
 32395 the *times* argument or with both *tv\_nsec* fields set to the special value **UTIME\_NOW**. Only a  
 32396 process with the effective user ID equal to the user ID of the file or with appropriate privileges  
 32397 may use *futimens()* or *utimensat()* with a non-null *times* argument that does not have both  
 32398 *tv\_nsec* fields set to **UTIME\_NOW** and does not have both *tv\_nsec* fields set to **UTIME\_OMIT**. If  
 32399 both *tv\_nsec* fields are set to **UTIME\_OMIT**, no ownership or permissions check shall be  
 32400 performed for the file, but other error conditions may still be detected (including **[EACCESS]**  
 32401 errors related to the path prefix).

32402 Values for the *flag* argument of *utimensat()* are constructed by a bitwise-inclusive OR of flags  
 32403 from the following list, defined in **<fcntl.h>**:

- 32404 AT\_SYMLINK\_NOFOLLOW  
 32405 If *path* names a symbolic link, then the access and modification times of the symbolic link  
 32406 are changed.
- 32407 Upon completion, *futimens()* and *utimensat()* shall mark the last file status change timestamp for  
 32408 update.
- 32409 The *utimes()* function shall be equivalent to the *utimensat()* function with the special value  
 32410 AT\_FDCWD as the *fd* argument and the *flag* argument set to zero, except that the *times* argument  
 32411 is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond,  
 32412 not nanosecond, and rounding towards the nearest second may occur.
- 32413 **RETURN VALUE**
- 32414 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 32415 return -1 and set *errno* to indicate the error. If -1 is returned, the file times shall not be affected.
- 32416 **ERRORS**
- 32417 These functions shall fail if:
- 32418 [EACCES] The *times* argument is a null pointer, or both *tv\_nsec* values are UTIME\_NOW,  
 32419 and the effective user ID of the process does not match the owner of the file  
 32420 and write access is denied.
- 32421 [EINVAL] Either of the *times* argument structures specified a *tv\_nsec* value that was  
 32422 neither UTIME\_NOW nor UTIME\_OMIT, and was a value less than zero or  
 32423 greater than or equal to 1 000 million.
- 32424 [EINVAL] A new file timestamp would be a value whose *tv\_sec* component is not a value  
 32425 supported by the file system.
- 32426 [EPERM] The *times* argument is not a null pointer, does not have both *tv\_nsec* fields set  
 32427 to UTIME\_NOW, does not have both *tv\_nsec* fields set to UTIME\_OMIT, the  
 32428 calling process' effective user ID has write access to the file but does not match  
 32429 the owner of the file, and the calling process does not have appropriate  
 32430 privileges.
- 32431 [EROFS] The file system containing the file is read-only.
- 32432 The *futimens()* function shall fail if:
- 32433 [EBADF] The *fd* argument is not a valid file descriptor.
- 32434 The *utimensat()* function shall fail if:
- 32435 [EACCES] *fd* was not opened with O\_SEARCH and the permissions of the directory  
 32436 underlying *fd* do not permit directory searches.
- 32437 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
 32438 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.
- 32439 The *utimensat()* and *utimes()* functions shall fail if:
- 32440 [EACCES] Search permission is denied by a component of the path prefix.
- 32441 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 32442 argument.
- 32443 [ENAMETOOLONG]  
 32444 The length of a component of a pathname is longer than {NAME\_MAX}.

**futimens()**

System Interfaces

- 32445 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 32446 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
32447 contains at least one non-`<slash>` character and ends with one or more trailing  
32448 `<slash>` characters and the last pathname component names an existing file  
32449 that is neither a directory nor a symbolic link to a directory.
- 32450 The *utimensat()* and *utimes()* functions may fail if:
- 32451 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
32452 resolution of the *path* argument.
- 32453 [ENAMETOOLONG]  
32454 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
32455 symbolic link produced an intermediate result with a length that exceeds  
32456 {PATH\_MAX}.
- 32457 The *utimensat()* function may fail if:
- 32458 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
32459 file descriptor associated with a directory.

**EXAMPLES**

32460 None.

**APPLICATION USAGE**

32463 None.

**RATIONALE**

32465 The purpose of the *utimensat()* function is to set the access and modification time of files in  
32466 directories other than the current working directory without exposure to race conditions. Any  
32467 part of the path of a file could be changed in parallel to a call to *utimes()*, resulting in unspecified  
32468 behavior. By opening a file descriptor for the target directory and using the *utimensat()* function  
32469 it can be guaranteed that the changed file is located relative to the desired directory.

32470 The standard developers considered including a special case for the permissions required by  
32471 *utimensat()* when one *tv\_nsec* field is UTIME\_NOW and the other is UTIME\_OMIT. One  
32472 possibility would be to include this case in with the cases where *times* is a null pointer or both  
32473 fields are UTIME\_NOW, where the call is allowed if the process has write permission for the file.  
32474 However, associating write permission with an update to just the last data access timestamp  
32475 (which is normally updated by *read()*) did not seem appropriate. The other possibility would be  
32476 to specify that this one case is allowed if the process has read permission, but this was felt to be  
32477 too great a departure from the *utime()* and *utimes()* functions on which *utimensat()* is based. If  
32478 an application needs to set the last data access timestamp to the current time for a file on which  
32479 it has read permission but is not the owner, it can do so by opening the file, reading one or more  
32480 bytes (or reading a directory entry, if the file is a directory), and then closing it.

**FUTURE DIRECTIONS**

32482 None.

**SEE ALSO**32484 *read()*, *utime()*32485 XBD `<fcntl.h>`, `<sys/stat.h>`, `<sys/time.h>`

32486 **CHANGE HISTORY**

32487 First released in Issue 4, Version 2.

32488 **Issue 5**

32489 Moved from X/OPEN UNIX extension to BASE.

32490 **Issue 6**

32491 This function is marked LEGACY.

32492 The normative text is updated to avoid use of the term “must” for application requirements.

32493 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
32494 [ELOOP] error condition is added.32495 **Issue 7**

32496 Austin Group Interpretation 1003.1-2001 #143 is applied.

32497 The LEGACY marking is removed.

32498 The *utimensat()* function (renamed from *futimesat()*) is added from The Open Group Technical  
32499 Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by  
32500 adding a *flag* argument.32501 The *futimens()* function is added.

32502 Changes are made related to support for finegrained timestamps.

32503 Changes are made to allow a directory to be opened for searching.

32504 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
32505 pathname exists but is not a directory or a symbolic link to a directory.

**fwide()**32506 **NAME**32507 `fwide` — set stream orientation32508 **SYNOPSIS**32509 `#include <stdio.h>`32510 `#include <wchar.h>`32511 `int fwide(FILE *stream, int mode);`32512 **DESCRIPTION**32513 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
32514 conflict between the requirements described here and the ISO C standard is unintentional. This  
32515 volume of POSIX.1-2008 defers to the ISO C standard.32516 The `fwide()` function shall determine the orientation of the stream pointed to by `stream`. If `mode` is  
32517 greater than zero, the function first attempts to make the stream wide-oriented. If `mode` is less  
32518 than zero, the function first attempts to make the stream byte-oriented. Otherwise, `mode` is zero  
32519 and the function does not alter the orientation of the stream.32520 If the orientation of the stream has already been determined, `fwide()` shall not change it.32521 **CX** Since no return value is reserved to indicate an error, an application wishing to check for error  
32522 situations should set `errno` to 0, then call `fwide()`, then check `errno`, and if it is non-zero, assume  
32523 an error has occurred.32524 **RETURN VALUE**32525 The `fwide()` function shall return a value greater than zero if, after the call, the stream has wide-  
32526 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no  
32527 orientation.32528 **ERRORS**32529 The `fwide()` function may fail if:32530 **CX** [EBADF] The `stream` argument is not a valid stream.32531 **EXAMPLES**

32532 None.

32533 **APPLICATION USAGE**32534 A call to `fwide()` with `mode` set to zero can be used to determine the current orientation of a  
32535 stream.32536 **RATIONALE**

32537 None.

32538 **FUTURE DIRECTIONS**

32539 None.

32540 **SEE ALSO**32541 XBD `<stdio.h>`, `<wchar.h>`32542 **CHANGE HISTORY**32543 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
32544 (E).32545 **Issue 6**

32546 Extensions beyond the ISO C standard are marked.

32547 **NAME**

32548 fwprintf, swprintf, wprintf — print formatted wide-character output

32549 **SYNOPSIS**

```

32550 #include <stdio.h>
32551 #include <wchar.h>

32552 int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
32553 int swprintf(wchar_t *restrict ws, size_t n,
32554             const wchar_t *restrict format, ...);
32555 int wprintf(const wchar_t *restrict format, ...);

```

32556 **DESCRIPTION**

32557 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32558 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32559 volume of POSIX.1-2008 defers to the ISO C standard.

32560 The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function  
 32561 shall place output on the standard output stream *stdout*. The *swprintf()* function shall place  
 32562 output followed by the null wide character in consecutive wide characters starting at *\*ws*; no  
 32563 more than *n* wide characters shall be written, including a terminating null wide character, which  
 32564 is always added (unless *n* is zero).

32565 Each of these functions shall convert, format, and print its arguments under control of the *format*  
 32566 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*,  
 32567 which are simply copied to the output stream, and *conversion specifications*, each of which results  
 32568 in the fetching of zero or more arguments. The results are undefined if there are insufficient  
 32569 arguments for the *format*. If the *format* is exhausted while arguments remain, the excess  
 32570 arguments are evaluated but are otherwise ignored.

32571 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 32572 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 32573 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range  
 32574 [1,{NL\_ARGMAX}], giving the position of the argument in the argument list. This feature  
 32575 provides for the definition of *format* wide-character strings that select arguments in an order  
 32576 appropriate to specific languages (see the EXAMPLES section).

32577 The *format* can contain either numbered argument specifications (that is, "%n\$" and "\*m\$"), or  
 32578 unnumbered argument conversion specifications (that is, % and \*), but not both. The only  
 32579 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered  
 32580 and unnumbered argument specifications in a *format* wide-character string are undefined. When  
 32581 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 32582 leading arguments, from the first to the (*N*-1)th, are specified in the *format* wide-character string.

32583 In *format* wide-character strings containing the "%n\$" form of conversion specification,  
 32584 numbered arguments in the argument list can be referenced from the *format* wide-character  
 32585 string as many times as required.

32586 In *format* wide-character strings containing the % form of conversion specification, each  
 32587 argument in the argument list shall be used exactly once.

32588 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character  
 32589 in the output string, output as a wide-character value. The radix character is defined in the  
 32590 locale of the process (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 32591 radix character is not defined, the radix character shall default to a <period> ('.').

## fwprintf()

- 32592 CX Each conversion specification is introduced by the ' % ' wide character or by the wide-character  
32593 sequence "%n\$", after which the following appear in sequence:
- 32594 • Zero or more *flags* (in any order), which modify the meaning of the conversion  
32595 specification.
  - 32596 • An optional minimum *field width*. If the converted value has fewer wide characters than  
32597 the field width, it shall be padded with <space> characters by default on the left; it shall be  
32598 padded on the right, if the left-adjustment flag (' - '), described below, is given to the field  
32599 width. The field width takes the form of an <asterisk> (' \* '), described below, or a decimal  
32600 integer.
  - 32601 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,  
32602 x, and X conversion specifiers; the number of digits to appear after the radix character for  
32603 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for  
32604 the g and G conversion specifiers; or the maximum number of wide characters to be  
32605 printed from a string in the s conversion specifiers. The precision takes the form of a  
32606 <period> (' . ') followed either by an <asterisk> (' \* '), described below, or an optional  
32607 decimal digit string, where a null digit string is treated as 0. If a precision appears with any  
32608 other conversion wide character, the behavior is undefined.
  - 32609 • An optional length modifier that specifies the size of the argument.
  - 32610 • A *conversion specifier* wide character that indicates the type of conversion to be applied.
- 32611 A field width, or precision, or both, may be indicated by an <asterisk> (' \* '). In this case an  
32612 argument of type **int** supplies the field width or precision. Applications shall ensure that  
32613 arguments specifying field width, or precision, or both appear in that order before the argument,  
32614 if any, to be converted. A negative field width is taken as a ' - ' flag followed by a positive field  
32615 CX width. A negative precision is taken as if the precision were omitted. In *format* wide-character  
32616 strings containing the "%n\$" form of a conversion specification, a field width or precision may  
32617 be indicated by the sequence "\*m\$", where *m* is a decimal integer in the range  
32618 [1,{NL\_ARGMAX}] giving the position in the argument list (after the *format* argument) of an  
32619 integer argument containing the field width or precision, for example:
- ```
32620 wprintf(L"%1$d:%2$. *3$d:%4$. *3$d\n", hour, min, precision, sec);
```
- 32621 The flag wide characters and their meanings are:
- 32622 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,  
32623 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For  
32624 other conversions, the behavior is undefined. The numeric grouping wide character is  
32625 used.
  - 32626 - The result of the conversion shall be left-justified within the field. The conversion shall  
32627 be right-justified if this flag is not specified.
  - 32628 + The result of a signed conversion shall always begin with a sign (' + ' or ' - '). The  
32629 conversion shall begin with a sign only when a negative value is converted if this flag is  
32630 not specified.
  - 32631 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion  
32632 results in no wide characters, a <space> shall be prefixed to the result. This means that  
32633 if the <space> and ' + ' flags both appear, the <space> flag shall be ignored.
  - 32634 # Specifies that the value is to be converted to an alternative form. For o conversion, it  
32635 increases the precision (if necessary) to force the first digit of the result to be 0. For x or  
32636 X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,

32637		E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.
32638		
32639		
32640		
32641		
32642	0	For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and <apostrophe> flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined.
32643		
32644		
32645		
32646	CX	
32647		
32648		
32649		The length modifiers and their meanings are:
32650	hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>signed char</b> or <b>unsigned char</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>signed char</b> or <b>unsigned char</b> before printing); or that a following n conversion specifier applies to a pointer to a <b>signed char</b> argument.
32651		
32652		
32653		
32654		
32655	h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>short</b> or <b>unsigned short</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>short</b> or <b>unsigned short</b> before printing); or that a following n conversion specifier applies to a pointer to a <b>short</b> argument.
32656		
32657		
32658		
32659		
32660	l (ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long</b> or <b>unsigned long</b> argument; that a following n conversion specifier applies to a pointer to a <b>long</b> argument; that a following c conversion specifier applies to a <b>wint_t</b> argument; that a following s conversion specifier applies to a pointer to a <b>wchar_t</b> argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.
32661		
32662		
32663		
32664		
32665	ll (ell-ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long long</b> or <b>unsigned long long</b> argument; or that a following n conversion specifier applies to a pointer to a <b>long long</b> argument.
32666		
32667		
32668		
32669	j	Specifies that a following d, i, o, u, x, or X conversion specifier applies to an <b>intmax_t</b> or <b>uintmax_t</b> argument; or that a following n conversion specifier applies to a pointer to an <b>intmax_t</b> argument.
32670		
32671		
32672	z	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>size_t</b> or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to a <b>size_t</b> argument.
32673		
32674		
32675	t	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type argument; or that a following n conversion specifier applies to a pointer to a <b>ptrdiff_t</b> argument.
32676		
32677		
32678	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a <b>long double</b> argument.
32679		

## fwprintf()

32680		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
32681		
32682		The conversion specifiers and their meanings are:
32683	d, i	The <b>int</b> argument shall be converted to a signed decimal in the style "[ - ] dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
32684		
32685		
32686		
32687		
32688	o	The <b>unsigned</b> argument shall be converted to unsigned octal format in the style " dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
32689		
32690		
32691		
32692		
32693	u	The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style " dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
32694		
32695		
32696		
32697		
32698	x	The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style " dddd"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
32699		
32700		
32701		
32702		
32703	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".
32704		
32705	f, F	The <b>double</b> argument shall be converted to decimal notation in the style "[ - ] ddd.ddd", where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.
32706		
32707		
32708		
32709		
32710		
32711		A <b>double</b> argument representing an infinity shall be converted in one of the styles "[ - ] inf" or "[ - ] infinity"; which style is implementation-defined. A <b>double</b> argument representing a NaN shall be converted in one of the styles "[ - ] nan" or "[ - ] nan ( n-char-sequence )"; which style, and the meaning of any n-char-sequence, is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.
32712		
32713		
32714		
32715		
32716		
32717	e, E	The <b>double</b> argument shall be converted in the style "[ - ] d. dde±dd", where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The E conversion wide character shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.
32718		
32719		
32720		
32721		
32722		
32723		
32724		
32725		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
32726		

32727 32728 32729 32730 32731	g, G	The <b>double</b> argument representing a floating-point number shall be converted in the style <code>f</code> or <code>e</code> (or in the style <code>F</code> or <code>E</code> in the case of a <code>G</code> conversion specifier), depending on the value converted and the precision. Let <code>P</code> equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style <code>E</code> would have an exponent of <code>X</code> :
32732		— If $P > X \geq -4$ , the conversion shall be with style <code>f</code> (or <code>F</code> ) and precision $P - (X + 1)$ .
32733		— Otherwise, the conversion shall be with style <code>e</code> (or <code>E</code> ) and precision $P - 1$ .
32734 32735 32736		Finally, unless the <code>'#'</code> flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.
32737 32738		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
32739 32740 32741 32742 32743 32744 32745 32746 32747 32748 32749 32750 32751 32752	a, A	A <b>double</b> argument representing a floating-point number shall be converted in the style <code>"[-]0xh.hhhhp±d"</code> , where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and <code>FLT_RADIX</code> is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision shall be sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be omitted; if the precision is zero and the <code>'#'</code> flag is not specified, no decimal-point wide character shall appear. The letters <code>"abcdef"</code> are used for a conversion and the letters <code>"ABCDEF"</code> for A conversion. The A conversion specifier produces a number with <code>'X'</code> and <code>'P'</code> instead of <code>'x'</code> and <code>'p'</code> . The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.
32753 32754		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
32755 32756 32757	c	If no <code>l</code> (ell) qualifier is present, the <b>int</b> argument shall be converted to a wide character as if by calling the <code>btowc()</code> function and the resulting wide character shall be written. Otherwise, the <b>wint_t</b> argument shall be converted to <b>wchar_t</b> , and written.
32758 32759 32760 32761 32762 32763 32764 32765 32766	s	If no <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <code>mbtowc()</code> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.
32767 32768 32769 32770 32771 32772		If an <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type <b>wchar_t</b> . Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.

**fwprintf()**

32773	p	The application shall ensure that the argument is a pointer to <b>void</b> . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
32774		
32775		
32776	n	The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <i>fwprintf()</i> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
32777		
32778		
32779		
32780		
32781	XSI	<b>C</b> Equivalent to <code>lc</code> .
32782	XSI	<b>S</b> Equivalent to <code>ls</code> .
32783	%	Output a <code>' %'</code> wide character; no argument shall be converted. The entire conversion specification shall be <code>%%</code> .
32784		
32785		If a conversion specification does not match one of the above forms, the behavior is undefined.
32786		In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <i>fwprintf()</i> and <i>wprintf()</i> shall be printed as if <i>fputwc()</i> had been called.
32787		
32788		
32789		
32790		For <i>a</i> and <i>A</i> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.
32791		
32792		
32793		
32794		For <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , and <i>G</i> conversion specifiers, if the number of significant decimal digits is at most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$ , both having <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string <i>D</i> should satisfy $L \leq D \leq U$ , with the extra stipulation that the error should have a correct sign for the current rounding direction.
32795		
32796		
32797		
32798		
32799		
32800		
32801		
32802	CX	The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of <i>fwprintf()</i> or <i>wprintf()</i> and the next successful completion of a call to <i>fflush()</i> or <i>fclose()</i> on the same stream, or a call to <i>exit()</i> or <i>abort()</i> .
32803		
32804		
32805		
32806		<b>RETURN VALUE</b>
32807		Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of <i>swprintf()</i> , or a negative
32808		value if an output error was encountered, and set <i>errno</i> to indicate the error.
32809	CX	
32810		If <i>n</i> or more wide characters were requested to be written, <i>swprintf()</i> shall return a negative
32811	CX	value, and set <i>errno</i> to indicate the error.
32812		<b>ERRORS</b>
32813		For the conditions under which <i>fwprintf()</i> and <i>wprintf()</i> fail and may fail, refer to <i>fputwc()</i> .
32814		In addition, all forms of <i>fwprintf()</i> shall fail if:
32815	CX	<b>[EILSEQ]</b> A wide-character code that does not correspond to a valid character has been
32816		detected.

32817 In addition, all forms of *fwprintf()* may fail if:

32818 CX [EINVAL] There are insufficient arguments.

32819 In addition, *fwprintf()* and *wprintf()* may fail if:

32820 CX [ENOMEM] Insufficient storage space is available.

32821 The *swprintf()* shall fail if:

32822 CX [EOVERFLOW] The value of *n* is greater than {INT\_MAX} or the number of bytes needed to  
32823 hold the output excluding the terminating null is greater than {INT\_MAX}.

#### 32824 EXAMPLES

32825 To print the language-independent date and time format, the following statement could be used:

```
32826 wprintf(format, weekday, month, day, hour, min);
```

32827 For American usage, *format* could be a pointer to the wide-character string:

```
32828 L"%s, %s %d, %d:%.2d\n"
```

32829 producing the message:

```
32830 Sunday, July 3, 10:02
```

32831 whereas for German usage, *format* could be a pointer to the wide-character string:

```
32832 L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

32833 producing the message:

```
32834 Sonntag, 3. Juli, 10:02
```

#### 32835 APPLICATION USAGE

32836 None.

#### 32837 RATIONALE

32838 None.

#### 32839 FUTURE DIRECTIONS

32840 None.

#### 32841 SEE ALSO

32842 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*

32843 XBD Chapter 7 (on page 135), `<stdio.h>`, `<wchar.h>`

#### 32844 CHANGE HISTORY

32845 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
32846 (E).

#### 32847 Issue 6

32848 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing  
32849 the case if *n* or more wide characters are requested to be written using *swprintf()*.

32850 The normative text is updated to avoid use of the term “must” for application requirements.

32851 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32852 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 32853 • The DESCRIPTION is updated.

**fwprintf()**

- 32854           • The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- 32855           • The `a`, `A`, and `F` conversion characters are added.
- 32856           • XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.
- 32857
- 32858           The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.
- 32859
- 32860           ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.
- 32861   **Issue 7**
- 32862           Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the `0` flag.
- 32863
- 32864           Austin Group Interpretation 1003.1-2001 #170 is applied.
- 32865           ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.
- 32866
- 32867           Functionality relating to the “`%n$`” form of conversion specification and the `<apostrophe>` flag is moved from the XSI option to the Base.
- 32868
- 32869           The [Eoverflow] error is added for `swprintf()`.
- 32870           Changes are made related to support for finegrained timestamps.

32871 **NAME**

32872 fwrite — binary output

32873 **SYNOPSIS**

32874 #include &lt;stdio.h&gt;

32875 size\_t fwrite(const void \*restrict ptr, size\_t size, size\_t nitems,  
32876 FILE \*restrict stream);32877 **DESCRIPTION**32878 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
32879 conflict between the requirements described here and the ISO C standard is unintentional. This  
32880 volume of POSIX.1-2008 defers to the ISO C standard.32881 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose  
32882 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be  
32883 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly  
32884 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by  
32885 the number of bytes successfully written. If an error occurs, the resulting value of the file-  
32886 position indicator for the stream is unspecified.32887 CX The last data modification and last file status change timestamps of the file shall be marked for  
32888 update between the successful execution of *fwrite()* and the next successful completion of a call  
32889 to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.32890 **RETURN VALUE**32891 The *fwrite()* function shall return the number of elements successfully written, which may be  
32892 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the  
32893 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for  
32894 CX the stream shall be set, and *errno* shall be set to indicate the error.32895 **ERRORS**32896 Refer to *fputc()*.32897 **EXAMPLES**

32898 None.

32899 **APPLICATION USAGE**32900 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
32901 are application-dependent, and possibly cannot be read using *fread()* by a different application  
32902 or by the same application on a different processor.32903 **RATIONALE**

32904 None.

32905 **FUTURE DIRECTIONS**

32906 None.

32907 **SEE ALSO**32908 *ferror()*, *fopen()*, *fprintf()*, *putc()*, *puts()*, *write()*

32909 XBD &lt;stdio.h&gt;

32910 **CHANGE HISTORY**

32911 First released in Issue 1. Derived from Issue 1 of the SVID.

**fwrite()**32912 **Issue 6**

32913 Extensions beyond the ISO C standard are marked.

32914 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32915 • The *fwrite()* prototype is updated.
- 32916 • The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

32917 **Issue 7**

32918 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

32919 **NAME**

32920 fwscanf, swscanf, wscanf — convert formatted wide-character input

32921 **SYNOPSIS**

32922 #include &lt;stdio.h&gt;

32923 #include &lt;wchar.h&gt;

32924 int fwscanf(FILE \*restrict stream, const wchar\_t \*restrict format, ...);

32925 int swscanf(const wchar\_t \*restrict ws,

32926 const wchar\_t \*restrict format, ...);

32927 int wscanf(const wchar\_t \*restrict format, ...);

32928 **DESCRIPTION**

32929 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32930 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32931 volume of POSIX.1-2008 defers to the ISO C standard.

32932 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read  
 32933 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character  
 32934 string *ws*. Each function reads wide characters, interprets them according to a format, and stores  
 32935 the results in its arguments. Each expects, as arguments, a control wide-character string *format*  
 32936 described below, and a set of *pointer* arguments indicating where the converted input should be  
 32937 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is  
 32938 exhausted while arguments remain, the excess arguments are evaluated but are otherwise  
 32939 ignored.

32940 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 32941 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 32942 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range  
 32943 [1,{NL\_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that  
 32944 select arguments in an order appropriate to specific languages. In *format* wide-character strings  
 32945 containing the "%n\$" form of conversion specifications, it is unspecified whether numbered  
 32946 arguments in the argument list can be referenced from the *format* wide-character string more  
 32947 than once.

32948 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 32949 two forms cannot normally be mixed within a single *format* wide-character string. The only  
 32950 exception to this is that %% or %\* can be mixed with the "%n\$" form. When numbered argument  
 32951 specifications are used, specifying the *N*th argument requires that all the leading arguments,  
 32952 from the first to the (*N*−1)th, are pointers.

32953 The *fwscanf()* function in all its forms allows for detection of a language-dependent radix  
 32954 character in the input string, encoded as a wide-character value. The radix character is defined  
 32955 in the locale of the process (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where  
 32956 the radix character is not defined, the radix character shall default to a <period> ('.').

32957 The *format* is a wide-character string composed of zero or more directives. Each directive is  
 32958 composed of one of the following: one or more white-space wide characters (<space>, <tab>,  
 32959 <newline>, <vertical-tab>, or <form-feed>); an ordinary wide character (neither '%' nor a  
 32960 white-space character); or a conversion specification.

32961 CX Each conversion specification is introduced by the '%' or by the character sequence "%n\$",  
 32962 after which the following appear in sequence:

- 32963 • An optional assignment-suppressing character '\*'.

**fwscanf()**

- 32964
- An optional non-zero decimal integer that specifies the maximum field width.
- 32965 CX
- An optional assignment-allocation character 'm'.
- 32966
- An optional length modifier that specifies the size of the receiving object.
- 32967
- A conversion specifier wide character that specifies the type of conversion to be applied.
- 32968
- The valid conversion specifiers are described below.
- 32969
- The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).
- 32970
- 32971
- 32972
- A directive composed of one or more white-space wide characters is executed by reading input until no more valid input can be read, or up to the first wide character which is not a white-space wide character, which remains unread.
- 32973
- 32974
- 32975
- A directive that is an ordinary wide character shall be executed as follows. The next wide character is read from the input and compared with the wide character that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a wide character from being read, the directive shall fail.
- 32976
- 32977
- 32978
- 32979
- 32980
- A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion wide character. A conversion specification is executed in the following steps.
- 32981
- 32982
- 32983
- Input white-space wide characters (as specified by *iswspace()*) shall be skipped, unless the conversion specification includes a *l*, *c*, or *n* conversion specifier.
- 32984
- 32985
- An item shall be read from the input unless the conversion specification includes an *n* conversion specifier wide character. An input item is defined as the longest sequence of input wide characters, not exceeding any specified field width, which is an initial subsequence of a matching sequence. The first wide character, if any, after the input item shall remain unread. If the length of the input item is zero, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.
- 32986
- 32987
- 32988
- 32989
- 32990
- 32991
- 32992
- Except in the case of a *%* conversion specifier, the input item (or, in the case of a *%n* conversion specification, the count of input wide characters) shall be converted to a type appropriate to the conversion wide character. If the input item is not a matching sequence, the execution of the conversion specification shall fail; this condition is a matching failure. Unless assignment suppression was indicated by a *'\*'*, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by *%*, or in the *n*th argument if introduced by the wide-character sequence "*%n\$*". If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.
- 32993
- 32994
- 32995
- 32996
- 32997
- 32998 CX
- 32999
- 33000
- 33001
- 33002 CX
- 33003
- 33004
- 33005
- 33006
- 33007
- 33008
- 33009
- The *%c*, *%s*, and *%[* conversion specifiers shall accept an optional assignment-allocation character 'm', which shall cause a memory buffer to be allocated to hold the wide-character string converted including a terminating null wide character. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer value that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. If the function returns EOF, any memory successfully allocated for

33010 parameters using assignment-allocation character 'm' by this call shall be freed before the  
33011 function returns.

33012 The length modifiers and their meanings are:

33013 hh Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33014 argument with type pointer to **signed char** or **unsigned char**.

33015 h Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33016 argument with type pointer to **short** or **unsigned short**.

33017 l (ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33018 argument with type pointer to **long** or **unsigned long**; that a following a, A, e, E, f, F,  
33019 g, or G conversion specifier applies to an argument with type pointer to **double**; or that  
33020 a following c, s, or [ conversion specifier applies to an argument with type pointer to  
33021 **wchar\_t**. If the 'm' assignment-allocation character is specified, the conversion  
33022 applies to an argument with the type pointer to a pointer to **wchar\_t**.

33023 ll (ell-ell)

33024 Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33025 argument with type pointer to **long long** or **unsigned long long**.

33026 j Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33027 argument with type pointer to **intmax\_t** or **uintmax\_t**.

33028 z Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33029 argument with type pointer to **size\_t** or the corresponding signed integer type.

33030 t Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
33031 argument with type pointer to **ptrdiff\_t** or the corresponding **unsigned** type.

33032 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an  
33033 argument with type pointer to **long double**.

33034 If a length modifier appears with any conversion specifier other than as specified above, the  
33035 behavior is undefined.

33036 The following conversion specifier wide characters are valid:

33037 d Matches an optionally signed decimal integer, whose format is the same as expected for  
33038 the subject sequence of *wcstol()* with the value 10 for the *base* argument. In the absence  
33039 of a size modifier, the application shall ensure that the corresponding argument is a  
33040 pointer to **int**.

33041 i Matches an optionally signed integer, whose format is the same as expected for the  
33042 subject sequence of *wcstol()* with 0 for the *base* argument. In the absence of a size  
33043 modifier, the application shall ensure that the corresponding argument is a pointer to  
33044 **int**.

33045 o Matches an optionally signed octal integer, whose format is the same as expected for  
33046 the subject sequence of *wcstoul()* with the value 8 for the *base* argument. In the absence  
33047 of a size modifier, the application shall ensure that the corresponding argument is a  
33048 pointer to **unsigned**.

33049 u Matches an optionally signed decimal integer, whose format is the same as expected for  
33050 the subject sequence of *wcstoul()* with the value 10 for the *base* argument. In the absence  
33051 of a size modifier, the application shall ensure that the corresponding argument is a  
33052 pointer to **unsigned**.

**fwscanf()**

33053	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
33054		
33055		In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .
33056		
33057	a, e, f, g	
33058		Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .
33059		
33060		
33061		
33062		If the <i>fwprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
33063		
33064		
33065	s	Matches a sequence of non-white-space wide characters. If no <b>l</b> (ell) qualifier is present, characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
33066		
33067		
33068		
33069		
33070		
33071	CX	
33072		
33073		
33074		If the <b>l</b> (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically. If the <b>l</b> (ell) qualifier is present and the 'm' assignment-allocation character is present, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
33075		
33076		
33077	CX	
33078		
33079		
33080	[	Matches a non-empty sequence of wide characters from a set of expected wide characters (the <i>scanset</i> ). If no <b>l</b> (ell) qualifier is present, wide characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
33081		
33082		
33083		
33084		
33085		
33086		
33087	CX	
33088		
33089		If an <b>l</b> (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character. If an <b>l</b> (ell) qualifier is present and the 'm' assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
33090		
33091		
33092	CX	
33093		
33094		
33095		The conversion specification includes all subsequent wide characters in the <i>format</i> string up to and including the matching <right-square-bracket> (' <b>]</b> '). The wide characters between the square brackets (the <i>scanlist</i> ) comprise the <i>scanset</i> , unless the wide character after the <left-square-bracket> is a <circumflex> (' <b>^</b> '), in which case the <i>scanset</i> contains all wide characters that do not appear in the <i>scanlist</i> between the <circumflex> and the <right-square-bracket>. If the conversion specification begins
33096		
33097		
33098		
33099		
33100		

33101			with "[ ]" or "[^]", the <right-square-bracket> is included in the scanlist and the
33102			next <right-square-bracket> is the matching <right-square-bracket> that ends the
33103			conversion specification; otherwise, the first <right-square-bracket> is the one that ends
33104			the conversion specification. If a '-' is in the scanlist and is not the first wide character,
33105			nor the second where the first wide character is a '^', nor the last wide character, the
33106			behavior is implementation-defined.
33107	c		Matches a sequence of wide characters of exactly the number specified by the field
33108			width (1 if no field width is present in the conversion specification).
33109			If no l (ell) length modifier is present, characters from the input field shall be converted
33110			as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by
33111			an <b>mbstate_t</b> object initialized to zero before the first wide character is converted. No
33112			null character is added. If the 'm' assignment-allocation character is not specified, the
33113			application shall ensure that the corresponding argument is a pointer to the initial
33114	CX		element of a character array large enough to accept the sequence. Otherwise, the
33115			application shall ensure that the corresponding argument is a pointer to a pointer to a
33116			<b>char</b> .
33117			No null wide character is added. If an l (ell) length modifier is present and the 'm'
33118			assignment-allocation character is not specified, the application shall ensure that the
33119			corresponding argument shall be a pointer to the initial element of an array of <b>wchar_t</b>
33120	CX		large enough to accept the sequence. If an l (ell) qualifier is present and the 'm'
33121			assignment-allocation character is specified, the application shall ensure that the
33122			corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
33123	p		Matches an implementation-defined set of sequences, which shall be the same as the set
33124			of sequences that is produced by the %p conversion specification of the corresponding
33125			<i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a
33126			pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-
33127			defined. If the input item is a value converted earlier during the same program
33128			execution, the pointer that results shall compare equal to that value; otherwise, the
33129			behavior of the %p conversion is undefined.
33130	n		No input is consumed. The application shall ensure that the corresponding argument is
33131			a pointer to the integer into which is to be written the number of wide characters read
33132			from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n
33133			conversion specification shall not increment the assignment count returned at the
33134			completion of execution of the function. No argument shall be converted, but one shall
33135			be consumed. If the conversion specification includes an assignment-suppressing wide
33136			character or a field width, the behavior is undefined.
33137	XSI	C	Equivalent to lc.
33138	XSI	S	Equivalent to ls.
33139		%	Matches a single '%' wide character; no conversion or assignment shall occur. The
33140			complete conversion specification shall be %%.
33141			If a conversion specification is invalid, the behavior is undefined.
33142			The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively,
33143			a, e, f, g, and x.
33144			If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before
33145			any wide characters matching the current conversion specification (except for %n) have been
33146			read (other than leading white-space, where permitted), execution of the current conversion

**fwscanf()**

33147 specification shall terminate with an input failure. Otherwise, unless execution of the current  
 33148 conversion specification is terminated with a matching failure, execution of the following  
 33149 conversion specification (if any) shall be terminated with an input failure.

33150 Reaching the end of the string in *swscanf()* shall be equivalent to encountering end-of-file for  
 33151 *fwscanf()*.

33152 If conversion terminates on a conflicting input, the offending input shall be left unread in the  
 33153 input. Any trailing white space (including <newline>) shall be left unread unless matched by a  
 33154 conversion specification. The success of literal matches and suppressed assignments is only  
 33155 directly determinable via the %n conversion specification.

33156 CX The *fwscanf()* and *wscanf()* functions may mark the last data access timestamp of the file  
 33157 associated with *stream* for update. The last data access timestamp shall be marked for update by  
 33158 the first successful execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*,  
 33159 *vwscanf()*, or *wscanf()* using *stream* that returns data not supplied by a prior call to *ungetwc()*.

**RETURN VALUE**

33160 Upon successful completion, these functions shall return the number of successfully matched  
 33161 and assigned input items; this number can be zero in the event of an early matching failure. If  
 33162 the input ends before the first matching failure or conversion, EOF shall be returned. If any  
 33163 error occurs, EOF shall be returned, and *errno* shall be set to indicate the error. If a read error  
 33164 CX occurs, the error indicator for the stream shall be set.  
 33165

**ERRORS**

33166 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.

33167 In addition, the *fwscanf()* function shall fail if:

33168 CX [EILSEQ] Input byte sequence does not form a valid character.

33169 [ENOMEM] Insufficient storage space is available.

33170 In addition, the *fwscanf()* function may fail if:

33171 CX [EINVAL] There are insufficient arguments.

**EXAMPLES**

33172 The call:

```
33173 int i, n; float x; char name[50];
33174 n = wscanf(L"%d%f%s", &i, &x, name);
```

33175 with the input line:

```
33176 25 54.32E-1 Hamster
```

33177 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
 33178 "Hamster".

33179 The call:

```
33180 int i; float x; char name[50];
33181 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

33182 with input:

```
33183 56789 0123 56a72
```

33184 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
 33185 *getchar()* shall return the character 'a'.

33188 **APPLICATION USAGE**

33189 In format strings containing the ' % ' form of conversion specifications, each argument in the  
33190 argument list is used exactly once.

33191 For functions that allocate memory as if by *malloc()*, the application should release such memory  
33192 when it is no longer required by a call to *free()*. For *fwscanf()*, this is memory allocated via use of  
33193 the ' m ' assignment-allocation character.

33194 **RATIONALE**

33195 None.

33196 **FUTURE DIRECTIONS**

33197 None.

33198 **SEE ALSO**

33199 *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*

33200 XBD Chapter 7 (on page 135), `<stdio.h>`, `<wchar.h>`

33201 **CHANGE HISTORY**

33202 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
33203 (E).

33204 **Issue 6**

33205 The normative text is updated to avoid use of the term "must" for application requirements.

33206 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 33207 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 33208 • The DESCRIPTION is updated.
- 33209 • The hh, ll, j, t, and z length modifiers are added.
- 33210 • The a, A, and F conversion characters are added.

33211 The DESCRIPTION is updated to use the terms "conversion specifier" and "conversion  
33212 specification" consistently.

33213 **Issue 7**

33214 Austin Group Interpretation 1003.1-2001 #170 is applied.

33215 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ' m '.

33216 Functionality relating to the "%n\$" form of conversion specification is moved from the XSI  
33217 option to the Base.

33218 Changes are made related to support for finegrained timestamps.

33219 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
33220 *malloc()*.

**gai\_strerror()**33221 **NAME**33222        **gai\_strerror** — address and name information error description33223 **SYNOPSIS**

33224        #include &lt;netdb.h&gt;

33225        const char \*gai\_strerror(int *ecode*);33226 **DESCRIPTION**33227        The *gai\_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*  
33228        and *getnameinfo()* functions listed in the <netdb.h> header.33229        When the *ecode* argument is one of the following values listed in the <netdb.h> header:

33230	[EAL_AGAIN]	[EAL_NONAME]
33231	[EAL_BADFLAGS]	[EAL_OVERFLOW]
33232	[EAL_FAIL]	[EAL_SERVICE]
33233	[EAL_FAMILY]	[EAL_SOCKETTYPE]
33234	[EAL_MEMORY]	[EAL_SYSTEM]

33235        the function return value shall point to a string describing the error. If the argument is not one  
33236        of those values, the function shall return a pointer to a string whose contents indicate an  
33237        unknown error.

33238 **RETURN VALUE**33239        Upon successful completion, *gai\_strerror()* shall return a pointer to an implementation-defined  
33240        string.33241 **ERRORS**

33242        No errors are defined.

33243 **EXAMPLES**

33244        None.

33245 **APPLICATION USAGE**

33246        None.

33247 **RATIONALE**

33248        None.

33249 **FUTURE DIRECTIONS**

33250        None.

33251 **SEE ALSO**33252        [freeaddrinfo\(\)](#)

33253        XBD &lt;netdb.h&gt;

33254 **CHANGE HISTORY**

33255        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

33256        The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from  
33257        **char \*** to **const char \***. This is for coordination with the IPnG Working Group.

33258        IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the  
33259        [EAL\_OVERFLOW] error code.

*System Interfaces***getaddrinfo()**33260 **NAME**33261 `getaddrinfo` — get address information33262 **SYNOPSIS**33263 `#include <sys/socket.h>`33264 `#include <netdb.h>`

```
33265 int getaddrinfo(const char *restrict nodename,  
33266                const char *restrict servname,  
33267                const struct addrinfo *restrict hints,  
33268                struct addrinfo **restrict res);
```

33269 **DESCRIPTION**33270 Refer to *freeaddrinfo()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getc()**33271 **NAME**33272            **getc** — get a byte from a stream33273 **SYNOPSIS**

33274            #include &lt;stdio.h&gt;

33275            int getc(FILE \*stream);

33276 **DESCRIPTION**33277 **CX**            The functionality described on this reference page is aligned with the ISO C standard. Any  
33278 conflict between the requirements described here and the ISO C standard is unintentional. This  
33279 volume of POSIX.1-2008 defers to the ISO C standard.33280            The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it  
33281 may evaluate *stream* more than once, so the argument should never be an expression with side-  
33282 effects.33283 **RETURN VALUE**33284            Refer to *fgetc()*.33285 **ERRORS**33286            Refer to *fgetc()*.33287 **EXAMPLES**

33288            None.

33289 **APPLICATION USAGE**33290            If the integer value returned by *getc()* is stored into a variable of type **char** and then compared  
33291 against the integer constant EOF, the comparison may never succeed, because sign-extension of  
33292 a variable of type **char** on widening to integer is implementation-defined.33293            Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with  
33294 side-effects. In particular, *getc(\*f++)* does not necessarily work as expected. Therefore, use of this  
33295 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.33296 **RATIONALE**

33297            None.

33298 **FUTURE DIRECTIONS**

33299            None.

33300 **SEE ALSO**33301            *fgetc()*

33302            XBD &lt;stdio.h&gt;

33303 **CHANGE HISTORY**

33304            First released in Issue 1. Derived from Issue 1 of the SVID.

33305 **NAME**

33306 `getc_unlocked`, `getchar_unlocked`, `putc_unlocked`, `putchar_unlocked` — `stdio` with explicit client  
 33307 locking

33308 **SYNOPSIS**

```
33309 CX #include <stdio.h>
33310
33310 int getc_unlocked(FILE *stream);
33311 int getchar_unlocked(void);
33312 int putc_unlocked(int c, FILE *stream);
33313 int putchar_unlocked(int c);
```

33314 **DESCRIPTION**

33315 Versions of the functions `getc()`, `getchar()`, `putc()`, and `putchar()` respectively named  
 33316 `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` shall be provided  
 33317 which are functionally equivalent to the original versions, with the exception that they are not  
 33318 required to be implemented in a thread-safe manner. They may only safely be used within a  
 33319 scope protected by `flockfile()` (or `ftrylockfile()`) and `funlockfile()`. These functions may safely be  
 33320 used in a multi-threaded program if and only if they are called while the invoking thread owns  
 33321 the (FILE \*) object, as is the case after a successful call to the `flockfile()` or `ftrylockfile()` functions.

33322 **RETURN VALUE**

33323 See `getc()`, `getchar()`, `putc()`, and `putchar()`.

33324 **ERRORS**

33325 See `getc()`, `getchar()`, `putc()`, and `putchar()`.

33326 **EXAMPLES**

33327 None.

33328 **APPLICATION USAGE**

33329 Since they may be implemented as macros, `getc_unlocked()` and `putc_unlocked()` may treat  
 33330 incorrectly a `stream` argument with side-effects. In particular, `getc_unlocked(*f++)` and  
 33331 `putc_unlocked(*f++)` do not necessarily work as expected. Therefore, use of these functions in  
 33332 such situations should be preceded by the following statement as appropriate:

```
33333 #undef getc_unlocked
33334 #undef putc_unlocked
```

33335 **RATIONALE**

33336 Some I/O functions are typically implemented as macros for performance reasons (for example,  
 33337 `putc()` and `getc()`). For safety, they need to be synchronized, but it is often too expensive to  
 33338 synchronize on every character. Nevertheless, it was felt that the safety concerns were more  
 33339 important; consequently, the `getc()`, `getchar()`, `putc()`, and `putchar()` functions are required to be  
 33340 thread-safe. However, unlocked versions are also provided with names that clearly indicate the  
 33341 unsafe nature of their operation but can be used to exploit their higher performance. These  
 33342 unlocked versions can be safely used only within explicitly locked program regions, using  
 33343 exported locking primitives. In particular, a sequence such as:

```
33344 flockfile(fileptr);
33345 putc_unlocked('1', fileptr);
33346 putc_unlocked('\n', fileptr);
33347 fprintf(fileptr, "Line 2\n");
33348 funlockfile(fileptr);
```

**getc\_unlocked()**

33349 is permissible, and results in the text sequence:

33350 1  
33351 Line 2

33352 being printed without being interspersed with output from other threads.

33353 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the  
33354 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still  
33355 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.  
33356 Choosing the safe bindings as the default, at least, results in correct code and maintains the  
33357 “atomicity at the function” invariant. To do otherwise would introduce gratuitous  
33358 synchronization errors into converted code. Other routines that modify the *stdio* (FILE \*)  
33359 structures or buffers are also safely synchronized.

33360 Note that there is no need for functions of the form *getc\_locked()*, *putc\_locked()*, and so on, since  
33361 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test  
33362 macro to switch a macro definition of *getc()* between *getc\_locked()* and *getc\_unlocked()*, since the  
33363 ISO C standard requires an actual function to exist, a function whose behavior could not be  
33364 changed by the feature test macro. Also, providing both the *xxx\_locked()* and *xxx\_unlocked()*  
33365 forms leads to the confusion of whether the suffix describes the behavior of the function or the  
33366 circumstances under which it should be used.

33367 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are  
33368 provided to allow the user to delineate a sequence of I/O statements that are executed  
33369 synchronously.

33370 The *ungetc()* function is infrequently called relative to the other functions/macros so no  
33371 unlocked variation is needed.

#### 33372 FUTURE DIRECTIONS

33373 None.

#### 33374 SEE ALSO

33375 *getc()*, *getchar()*, *putc()*, *putchar()*

33376 XBD <stdio.h>

#### 33377 CHANGE HISTORY

33378 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 33379 Issue 6

33380 These functions are marked as part of the Thread-Safe Functions option.

33381 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing  
33382 how applications should be written to avoid the case when the functions are implemented as  
33383 macros.

#### 33384 Issue 7

33385 The *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* functions are  
33386 moved from the Thread-Safe Functions option to the Base.

33387 **NAME**33388            *getchar* — get a byte from a *stdin* stream33389 **SYNOPSIS**

33390            #include &lt;stdio.h&gt;

33391            int *getchar*(void);33392 **DESCRIPTION**33393 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
33394 conflict between the requirements described here and the ISO C standard is unintentional. This  
33395 volume of POSIX.1-2008 defers to the ISO C standard.33396            The *getchar*() function shall be equivalent to *getc(stdin)*.33397 **RETURN VALUE**33398            Refer to *fgetc*().33399 **ERRORS**33400            Refer to *fgetc*().33401 **EXAMPLES**

33402            None.

33403 **APPLICATION USAGE**33404            If the integer value returned by *getchar*() is stored into a variable of type **char** and then  
33405 compared against the integer constant EOF, the comparison may never succeed, because sign-  
33406 extension of a variable of type **char** on widening to integer is implementation-defined.33407 **RATIONALE**

33408            None.

33409 **FUTURE DIRECTIONS**

33410            None.

33411 **SEE ALSO**33412            *getc*()

33413            XBD &lt;stdio.h&gt;

33414 **CHANGE HISTORY**

33415            First released in Issue 1. Derived from Issue 1 of the SVID.

**getchar\_unlocked()**

System Interfaces

33416 **NAME**

33417            getchar\_unlocked — stdio with explicit client locking

33418 **SYNOPSIS**

```
33419 CX        #include <stdio.h>  
33420            int getchar_unlocked(void);
```

33421 **DESCRIPTION**33422            Refer to *getc\_unlocked()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

33423 **NAME**33424 `getcwd` — get the pathname of the current working directory33425 **SYNOPSIS**

```
33426 #include <unistd.h>
33427 char *getcwd(char *buf, size_t size);
```

33428 **DESCRIPTION**

33429 The `getcwd()` function shall place an absolute pathname of the current working directory in the  
 33430 array pointed to by `buf`, and return `buf`. The pathname shall contain no components that are dot  
 33431 or dot-dot, or are symbolic links.

33432 If there are multiple pathnames that `getcwd()` could place in the array pointed to by `buf`, one  
 33433 beginning with a single <slash> character and one or more beginning with two <slash>  
 33434 characters, then `getcwd()` shall place the pathname beginning with a single <slash> character in  
 33435 the array. The pathname shall not contain any unnecessary <slash> characters after the leading  
 33436 one or two <slash> characters.

33437 The `size` argument is the size in bytes of the character array pointed to by the `buf` argument. If `buf`  
 33438 is a null pointer, the behavior of `getcwd()` is unspecified.

33439 **RETURN VALUE**

33440 Upon successful completion, `getcwd()` shall return the `buf` argument. Otherwise, `getcwd()` shall  
 33441 return a null pointer and set `errno` to indicate the error. The contents of the array pointed to by  
 33442 `buf` are then undefined.

33443 **ERRORS**

33444 The `getcwd()` function shall fail if:

33445	[EINVAL]	The <code>size</code> argument is 0.
33446	[ERANGE]	The <code>size</code> argument is greater than 0, but is smaller than the length of the string +1.

33448 The `getcwd()` function may fail if:

33449	[EACCES]	Search permission was denied for the current directory, or read or search permission was denied for a directory above the current directory in the file hierarchy.
33452	[ENOMEM]	Insufficient storage space is available.

33453 **EXAMPLES**

33454 The following example uses {PATH\_MAX} as the initial buffer size (unless it is indeterminate or  
 33455 very large), and calls `getcwd()` with progressively larger buffers until it does not give an  
 33456 [ERANGE] error.

```
33457 #include <stdlib.h>
33458 #include <errno.h>
33459 #include <unistd.h>
33460 ...
33461 long path_max;
33462 size_t size;
33463 char *buf;
33464 char *ptr;
33465 path_max = pathconf(".", _PC_PATH_MAX);
33466 if (path_max == -1)
```

**getcwd()**

```

33467         size = 1024;
33468     else if (path_max > 10240)
33469         size = 10240;
33470     else
33471         size = path_max;
33472     for (buf = ptr = NULL; ptr == NULL; size *= 2)
33473     {
33474         if ((buf = realloc(buf, size)) == NULL)
33475         {
33476             ... handle error ...
33477         }
33478         ptr = getcwd(buf, size);
33479         if (ptr == NULL && errno != ERANGE)
33480         {
33481             ... handle error ...
33482         }
33483     }
33484     free (buf);

```

**APPLICATION USAGE**

If the pathname obtained from *getcwd()* is longer than `{PATH_MAX}` bytes, it could produce an `[ENAMETOOLONG]` error if passed to *chdir()*. Therefore, in order to return to that directory it may be necessary to break the pathname into sections shorter than `{PATH_MAX}` bytes and call *chdir()* on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames). A simpler way to handle saving and restoring the working directory when it may be deeper than `{PATH_MAX}` bytes in the file hierarchy is to use a file descriptor and *fchdir()*, rather than *getcwd()* and *chdir()*. However, the two methods do have some differences. The *fchdir()* approach causes the program to restore a working directory even if it has been renamed in the meantime, whereas the *chdir()* approach restores to a directory with the same name as the original, even if the directories were renamed in the meantime. Since the *fchdir()* approach does not access parent directories, it can succeed when *getcwd()* would fail due to permissions problems. In applications conforming to earlier versions of this standard, it was not possible to use the *fchdir()* approach when the working directory is searchable but not readable, as the only way to open a directory was with `O_RDONLY`, whereas the *getcwd()* approach can succeed in this case.

**RATIONALE**

Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for the returned argument was considered. The advantage is that *getcwd()* knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the *free()* function to free the resulting object, or each use of *getcwd()* would further reduce the available memory. Finally, *getcwd()* is taken from the SVID where it has the two arguments used in this volume of POSIX.1-2008.

The older function *getwd()* was rejected for use in this context because it had only a buffer argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in conforming applications.

33515 Earlier implementations of `getcwd()` sometimes generated pathnames like  
 33516 `"../../../../subdirname"` internally, using them to explore the path of ancestor directories  
 33517 back to the root. If one of these internal pathnames exceeded `{PATH_MAX}` in length, the  
 33518 implementation could fail with `errno` set to `[ENAMETOOLONG]`. This is no longer allowed.

33519 If a program is operating in a directory where some (grand)parent directory does not permit  
 33520 reading, `getcwd()` may fail, as in most implementations it must read the directory to determine  
 33521 the name of the file. This can occur if search, but not read, permission is granted in an  
 33522 intermediate directory, or if the program is placed in that directory by some more privileged  
 33523 process (for example, login). Including the `[EACCES]` error condition makes the reporting of the  
 33524 error consistent and warns the application developer that `getcwd()` can fail for reasons beyond  
 33525 the control of the application developer or user. Some implementations can avoid this  
 33526 occurrence (for example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root  
 33527 process), thus the error was made optional. Since this volume of POSIX.1-2008 permits the  
 33528 addition of other errors, this would be a common addition and yet one that applications could  
 33529 not be expected to deal with without this addition.

#### 33530 FUTURE DIRECTIONS

33531 None.

#### 33532 SEE ALSO

33533 [`malloc\(\)`](#)

33534 XBD [`<unistd.h>`](#)

#### 33535 CHANGE HISTORY

33536 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 33537 Issue 6

33538 The following new requirements on POSIX implementations derive from alignment with the  
 33539 Single UNIX Specification:

- 33540 • The `[ENOMEM]` optional error condition is added.

#### 33541 Issue 7

33542 Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with  
 33543 the `pwd` utility, adding text to address the case where the current directory is deeper in the file  
 33544 hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames  
 33545 beginning with two `<slash>` characters.

**getdate()**33546 **NAME**33547 `getdate` — convert user format date and time33548 **SYNOPSIS**

```
33549 XSI #include <time.h>
33550 struct tm *getdate(const char *string);
```

33551 **DESCRIPTION**

33552 The `getdate()` function shall convert a string representation of a date or time into a broken-down  
 33553 time.

33554 The external variable or macro `getdate_err`, which has type **int**, is used by `getdate()` to return error  
 33555 values. It is unspecified whether `getdate_err` is a macro or an identifier declared with external  
 33556 linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order  
 33557 to access an actual object, or a program defines an identifier with the name `getdate_err`, the  
 33558 behavior is undefined.

33559 Templates are used to parse and interpret the input string. The templates are contained in a text  
 33560 file identified by the environment variable `DATEMSK`. The `DATEMSK` variable should be set to  
 33561 indicate the full pathname of the file that contains the templates. The first line in the template  
 33562 that matches the input specification is used for interpretation and conversion into the internal  
 33563 time format.

33564 The following conversion specifications shall be supported:

33565	<code>%%</code>	Equivalent to <code>%</code> .
33566	<code>%a</code>	Abbreviated weekday name.
33567	<code>%A</code>	Full weekday name.
33568	<code>%b</code>	Abbreviated month name.
33569	<code>%B</code>	Full month name.
33570	<code>%c</code>	Locale's appropriate date and time representation.
33571	<code>%C</code>	Century number [00,99]; leading zeros are permitted but not required.
33572	<code>%d</code>	Day of month [01,31]; the leading 0 is optional.
33573	<code>%D</code>	Date as <code>%m/%d/%y</code> .
33574	<code>%e</code>	Equivalent to <code>%d</code> .
33575	<code>%h</code>	Abbreviated month name.
33576	<code>%H</code>	Hour [00,23].
33577	<code>%I</code>	Hour [01,12].
33578	<code>%m</code>	Month number [01,12].
33579	<code>%M</code>	Minute [00,59].
33580	<code>%n</code>	Equivalent to <code>&lt;newline&gt;</code> .
33581	<code>%p</code>	Locale's equivalent of either AM or PM.
33582	<code>%r</code>	The locale's appropriate representation of time in AM and PM notation. In the POSIX 33583 locale, this shall be equivalent to <code>%I:%M:%S %p</code> .

33584	%R	Time as %H:%M.
33585	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap second data must come from some external source.
33586		
33587		
33588	%t	Equivalent to <tab>.
33589	%T	Time as %H:%M:%S.
33590	%w	Weekday number (Sunday = [0,6]).
33591	%x	Locale's appropriate date representation.
33592	%X	Locale's appropriate time representation.
33593	%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.
33594		
33595		
33596	<b>Note:</b>	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
33597		
33598		
33599	%Y	Year as "ccyy" (for example, 2001).
33600	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).
33601		
33602		
33603		
33604		The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.
33605		
33606		The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i> ).
33607		
33608		
33609		Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra white space in either the template file or in <i>string</i> shall be ignored.
33610		
33611		
33612		The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.
33613		
33614		The following rules apply for converting the input specification into the internal format:
33615		• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.
33616		
33617		
33618		• If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
33619		
33620		• If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
33621		
33622		
33623		• If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
33624		

**getdate()**

33625 • If no date is given, the hour chosen shall be the hour, starting with the current hour and  
33626 moving into the future, which first matches the named hour.

33627 If a conversion specification in the DATEMSK file does not correspond to one of the conversion  
33628 specifications above, the behavior is unspecified.

33629 The *getdate()* function need not be thread-safe.

**RETURN VALUE**

33631 Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall  
33632 return a null pointer and set *getdate\_err* to indicate the error.

**ERRORS**

33634 The *getdate()* function shall fail in the following cases, setting *getdate\_err* to the value shown in  
33635 the list below. Any changes to *errno* are unspecified.

- 33636 1. The DATEMSK environment variable is null or undefined.
- 33637 2. The template file cannot be opened for reading.
- 33638 3. Failed to get file status information.
- 33639 4. The template file is not a regular file.
- 33640 5. An I/O error is encountered while reading the template file.
- 33641 6. Memory allocation failed (not enough memory available).
- 33642 7. There is no line in the template that matches the input.
- 33643 8. Invalid input specification. For example, February 31; or a time is specified that cannot be  
33644 represented in a **time\_t** (representing the time in seconds since the Epoch).

**EXAMPLES**

- 33646 1. The following example shows the possible contents of a template:

```
33647 %m
33648 %A %B %d, %Y, %H:%M:%S
33649 %A
33650 %B
33651 %m/%d/%y %I %p
33652 %d, %m, %Y %H:%M
33653 at %A the %dst of %B in %Y
33654 run job at %I %p, %B %dnd
33655 %A den %d. %B %Y %H.%M Uhr
```

- 33656 2. The following are examples of valid input specifications for the template in Example 1:

```
33657 getdate("10/1/87 4 PM");
33658 getdate("Friday");
33659 getdate("Friday September 18, 1987, 10:30:30");
33660 getdate("24,9,1986 10:30");
33661 getdate("at monday the 1st of december in 1986");
33662 getdate("run job at 3 PM, december 2nd");
```

33663 If the *LC\_TIME* category is set to a German locale that includes *freitag* as a weekday name  
33664 and *oktober* as a month name, the following would be valid:

```
33665 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

- 33666 3. The following example shows how local date and time specification can be defined in the  
33667 template:

Invocation	Line in Template
33668 getdate ("11/27/86")	%m/%d/%y
33669 getdate ("27.11.86")	%d.%m.%y
33670 getdate ("86-11-27")	%y-%m-%d
33671 getdate ("Friday 12:00:00")	%A %H:%M:%S

- 33673 4. The following examples help to illustrate the above rules assuming that the current date  
33674 is Mon Sep 22 12:19:47 EDT 1986 and the *LC\_TIME* category is set to the default C locale:

Input	Line in Template	Date
33675 Mon	%a	Mon Sep 22 12:19:47 EDT 1986
33676 Sun	%a	Sun Sep 28 12:19:47 EDT 1986
33677 Fri	%a	Fri Sep 26 12:19:47 EDT 1986
33678 September	%B	Mon Sep 1 12:19:47 EDT 1986
33679 January	%B	Thu Jan 1 12:19:47 EST 1987
33680 December	%B	Mon Dec 1 12:19:47 EST 1986
33681 Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
33682 Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
33683 Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
33684 Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
33685 Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
33686 Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
33687 10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
33688 13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

#### 33690 APPLICATION USAGE

33691 Although historical versions of *getdate()* did not require that `<time.h>` declare the external  
33692 variable *getdate\_err*, this volume of POSIX.1-2008 does require it. The standard developers  
33693 encourage applications to remove declarations of *getdate\_err* and instead incorporate the  
33694 declaration by including `<time.h>`.

33695 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

#### 33696 RATIONALE

33697 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported  
33698 conversion specifiers and so the text regarding results being undefined is not a problem in that  
33699 case.

#### 33700 FUTURE DIRECTIONS

33701 None.

#### 33702 SEE ALSO

33703 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*

33704 XBD `<time.h>`

#### 33705 CHANGE HISTORY

33706 First released in Issue 4, Version 2.

**getdate()**

- 33707 **Issue 5**
- 33708 Moved from X/OPEN UNIX extension to BASE.
- 33709 The last paragraph of the DESCRIPTION is added.
- 33710 The %C conversion specification is added, and the exact meaning of the %y conversion specification is clarified in the DESCRIPTION.
- 33711
- 33712 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 33713 The %R conversion specification is changed to follow historical practice.
- 33714 **Issue 6**
- 33715 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time* functions.
- 33716
- 33717
- 33718 The description of %S is updated so that the valid range is [00,60] rather than [00,61].
- 33719 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors for consistency with other functions.
- 33720
- 33721 **Issue 7**
- 33722 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 33723 The description of the *getdate\_err* value is expanded.

33724 **NAME**33725 getdelim, getline — read a delimited record from *stream*33726 **SYNOPSIS**

```

33727 CX #include <stdio.h>
33728
33728 ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
33729                 int delimiter, FILE *restrict stream);
33730
33730 ssize_t getline(char **restrict lineptr, size_t *restrict n,
33731                FILE *restrict stream);

```

33732 **DESCRIPTION**

33733 The *getdelim()* function shall read from *stream* until it encounters a character matching the  
 33734 *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall  
 33735 ensure is a character representable as an **unsigned char** of equal value that terminates the read  
 33736 process. If the *delimiter* argument has any other value, the behavior is undefined.

33737 The application shall ensure that *\*lineptr* is a valid argument that could be passed to the *free()*  
 33738 function. If *\*n* is non-zero, the application shall ensure that *\*lineptr* either points to an object of  
 33739 size at least *\*n* bytes, or is a null pointer.

33740 The size of the object pointed to by *\*lineptr* shall be increased to fit the incoming line, if it isn't  
 33741 already large enough, including room for the delimiter and a terminating NUL. The characters  
 33742 read, including any delimiter, shall be stored in the string pointed to by the *lineptr* argument,  
 33743 and a terminating NUL added when the delimiter or end of file is encountered.

33744 The *getline()* function shall be equivalent to the *getdelim()* function with the *delimiter* character  
 33745 equal to the <newline> character.

33746 The *getdelim()* and *getline()* functions may mark the last data access timestamp of the file  
 33747 associated with *stream* for update. The last data access timestamp shall be marked for update by  
 33748 the first successful execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*,  
 33749 *getline()*, *gets()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

33750 **RETURN VALUE**

33751 Upon successful completion, the *getline()* and *getdelim()* functions shall return the number of  
 33752 characters written into the buffer, including the delimiter character if one was encountered  
 33753 before EOF, but excluding the terminating NUL character. If no characters were read, and the  
 33754 end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator  
 33755 for the stream shall be set and the function shall return  $-1$ . If an error occurs, the error indicator  
 33756 for the stream shall be set, and the function shall return  $-1$  and set *errno* to indicate the error.

33757 **ERRORS**

33758 For the conditions under which the *getdelim()* and *getline()* functions shall fail and may fail, refer  
 33759 to *fgetc()*.

33760 In addition, these functions shall fail if:

33761 [EINVAL] *lineptr* or *n* is a null pointer.

33762 [ENOMEM] Insufficient memory is available.

33763 These functions may fail if:

33764 [EOVERFLOW] More than {SSIZE\_MAX} characters were read without encountering the  
 33765 *delimiter* character.

**getdelim()**33766 **EXAMPLES**

```

33767     #include <stdio.h>
33768     #include <stdlib.h>

33769     int main(void)
33770     {
33771         FILE *fp;
33772         char *line = NULL;
33773         size_t len = 0;
33774         ssize_t read;
33775         fp = fopen("/etc/motd", "r");
33776         if (fp == NULL)
33777             exit(1);
33778         while ((read = getline(&line, &len, fp)) != -1) {
33779             printf("Retrieved line of length %zu :\n", read);
33780             printf("%s", line);
33781         }
33782         if (ferror(fp)) {
33783             /* handle error */
33784         }
33785         free(line);
33786         fclose(fp);
33787         return 0;
33788     }

```

33789 **APPLICATION USAGE**

33790 Setting *\*lineptr* to a null pointer and *\*n* to zero are allowed and a recommended way to start  
 33791 parsing a file.

33792 The *ferror()* or *feof()* functions should be used to distinguish between an error condition and an  
 33793 end-of-file condition.

33794 Although a NUL terminator is always supplied after the line, note that *strlen(\*lineptr)* will be  
 33795 smaller than the return value if the line contains embedded NUL characters.

33796 **RATIONALE**

33797 These functions are widely used to solve the problem that the *fgets()* function has with long  
 33798 lines. The functions automatically enlarge the target buffers if needed. These are especially  
 33799 useful since they reduce code needed for applications.

33800 **FUTURE DIRECTIONS**

33801 None.

33802 **SEE ALSO**

33803 *fgetc()*, *fgets()*, *free()*

33804 XBD <stdio.h>

33805 **CHANGE HISTORY**

33806 First released in Issue 7.

33807 **NAME**

33808           getegid — get the effective group ID

33809 **SYNOPSIS**

33810           #include &lt;unistd.h&gt;

33811           gid\_t getegid(void);

33812 **DESCRIPTION**33813           The *getegid()* function shall return the effective group ID of the calling process.33814 **RETURN VALUE**33815           The *getegid()* function shall always be successful and no return value is reserved to indicate an error.33817 **ERRORS**

33818           No errors are defined.

33819 **EXAMPLES**

33820           None.

33821 **APPLICATION USAGE**

33822           None.

33823 **RATIONALE**

33824           None.

33825 **FUTURE DIRECTIONS**

33826           None.

33827 **SEE ALSO**33828           *geteuid(), getgid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33829           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

33830 **CHANGE HISTORY**

33831           First released in Issue 1. Derived from Issue 1 of the SVID.

33832 **Issue 6**

33833           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

33834           The following new requirements on POSIX implementations derive from alignment with the  
33835           Single UNIX Specification:

- 33836           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
33837           required for conforming implementations of previous POSIX specifications, it was not  
33838           required for UNIX applications.

**getenv()**33839 **NAME**33840 `getenv` — get value of an environment variable33841 **SYNOPSIS**33842 `#include <stdlib.h>`33843 `char *getenv(const char *name);`33844 **DESCRIPTION**

33845 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 33846 conflict between the requirements described here and the ISO C standard is unintentional. This  
 33847 volume of POSIX.1-2008 defers to the ISO C standard.

33848 The `getenv()` function shall search the environment of the calling process (see XBD Chapter 8, on  
 33849 page 173) for the environment variable `name` if it exists and return a pointer to the value of the  
 33850 environment variable. If the specified environment variable cannot be found, a null pointer shall  
 33851 be returned. The application shall ensure that it does not modify the string pointed to by the  
 33852 `getenv()` function.

33853 CX The string pointed to may be overwritten by a subsequent call to `getenv()`, `setenv()`, `unsetenv()`,  
 33854 XSI or `putenv()` but shall not be overwritten by a call to any other function in this volume of  
 33855 POSIX.1-2008.

33856 CX If the application modifies `environ` or the pointers to which it points, the behavior of `getenv()` is  
 33857 undefined.

33858 The `getenv()` function need not be thread-safe.

33859 **RETURN VALUE**

33860 Upon successful completion, `getenv()` shall return a pointer to a string containing the *value* for  
 33861 the specified `name`. If the specified `name` cannot be found in the environment of the calling  
 33862 process, a null pointer shall be returned.

33863 **ERRORS**

33864 No errors are defined.

33865 **EXAMPLES**33866 **Getting the Value of an Environment Variable**

33867 The following example gets the value of the `HOME` environment variable.

```
33868 #include <stdlib.h>
33869 ...
33870 const char *name = "HOME";
33871 char *value;
33872 value = getenv(name);
```

33873 **APPLICATION USAGE**

33874 None.

33875 **RATIONALE**

33876 The `clearenv()` function was considered but rejected. The `putenv()` function has now been  
 33877 included for alignment with the Single UNIX Specification.

33878 The `getenv()` function is inherently not thread-safe because it returns a value pointing to static  
 33879 data.

33880 Conforming applications are required not to modify `environ` directly, but to use only the  
 33881 functions described here to manipulate the process environment as an abstract object. Thus, the

33882 implementation of the environment access functions has complete control over the data  
 33883 structure used to represent the environment (subject to the requirement that *environ* be  
 33884 maintained as a list of strings with embedded <equals-sign> characters for applications that  
 33885 wish to scan the environment). This constraint allows the implementation to properly manage  
 33886 the memory it allocates, either by using allocated storage for all variables (copying them on the  
 33887 first invocation of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in  
 33888 allocated space and which are not, via a separate table or some other means. This enables the  
 33889 implementation to free any allocated space used by strings (and perhaps the pointers to them)  
 33890 stored in *environ* when *unsetenv()* is called. A C runtime start-up procedure (that which invokes  
 33891 *main()* and perhaps initializes *environ*) can also initialize a flag indicating that none of the  
 33892 environment has yet been copied to allocated storage, or that the separate table has not yet been  
 33893 initialized.

33894 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate  
 33895 copy of the environment in a data structure that could be searched much more quickly (such as  
 33896 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when  
 33897 *setenv()* or *unsetenv()* is invoked.

33898 Performance of *getenv()* can be important for applications which have large numbers of  
 33899 environment variables. Typically, applications like this use the environment as a resource  
 33900 database of user-configurable parameters. The fact that these variables are in the user's shell  
 33901 environment usually means that any other program that uses environment variables (such as *ls*,  
 33902 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC\_ALL*, and so on) is  
 33903 similarly slowed down by the linear search through the variables.

33904 An implementation that maintains separate data structures, or even one that manages the  
 33905 memory it consumes, is not currently required as it was thought it would reduce consensus  
 33906 among implementors who do not want to change their historical implementations.

#### 33907 FUTURE DIRECTIONS

33908 A future version may add one or more functions to access and modify the environment in a  
 33909 thread-safe manner.

#### 33910 SEE ALSO

33911 *exec*, *putenv()*, *setenv()*, *unsetenv()*

33912 XBD Chapter 8 (on page 173), <stdlib.h>

#### 33913 CHANGE HISTORY

33914 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 33915 Issue 5

33916 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 33917 VALUE section.

33918 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

#### 33919 Issue 6

33920 The following changes were made to align with the IEEE P1003.1a draft standard:

- 33921 • References added to the new *setenv()* and *unsetenv()* functions.

33922 The normative text is updated to avoid use of the term "must" for application requirements.

#### 33923 Issue 7

33924 Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv()* may  
 33925 also cause the string to be overwritten.

33926 Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

**getenv()**

*System Interfaces*

33927

Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

33928 **NAME**

33929           geteuid — get the effective user ID

33930 **SYNOPSIS**

33931           #include &lt;unistd.h&gt;

33932           uid\_t geteuid(void);

33933 **DESCRIPTION**33934           The *geteuid()* function shall return the effective user ID of the calling process.33935 **RETURN VALUE**33936           The *geteuid()* function shall always be successful and no return value is reserved to indicate an  
33937 error.33938 **ERRORS**

33939           No errors are defined.

33940 **EXAMPLES**

33941           None.

33942 **APPLICATION USAGE**

33943           None.

33944 **RATIONALE**

33945           None.

33946 **FUTURE DIRECTIONS**

33947           None.

33948 **SEE ALSO**33949           *getegid(), getgid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33950           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

33951 **CHANGE HISTORY**

33952           First released in Issue 1. Derived from Issue 1 of the SVID.

33953 **Issue 6**

33954           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

33955           The following new requirements on POSIX implementations derive from alignment with the  
33956 Single UNIX Specification:

- 33957
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
33958 required for conforming implementations of previous POSIX specifications, it was not  
33959 required for UNIX applications.

**getgid()**33960 **NAME**

33961           getgid — get the real group ID

33962 **SYNOPSIS**

33963           #include &lt;unistd.h&gt;

33964           gid\_t getgid(void);

33965 **DESCRIPTION**33966           The *getgid()* function shall return the real group ID of the calling process.33967 **RETURN VALUE**33968           The *getgid()* function shall always be successful and no return value is reserved to indicate an  
33969           error.33970 **ERRORS**

33971           No errors are defined.

33972 **EXAMPLES**

33973           None.

33974 **APPLICATION USAGE**

33975           None.

33976 **RATIONALE**

33977           None.

33978 **FUTURE DIRECTIONS**

33979           None.

33980 **SEE ALSO**33981           *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

33982           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

33983 **CHANGE HISTORY**

33984           First released in Issue 1. Derived from Issue 1 of the SVID.

33985 **Issue 6**

33986           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

33987           The following new requirements on POSIX implementations derive from alignment with the  
33988           Single UNIX Specification:

- 33989
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 33990
- 
- 33991

*System Interfaces***getgrent()**33992 **NAME**

33993           getgrent — get the group database entry

33994 **SYNOPSIS**

```
33995 xSI       #include <grp.h>  
33996       struct group *getgrent(void);
```

33997 **DESCRIPTION**33998       Refer to *endgrent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getgrgid()**33999 **NAME**34000 `getgrgid, getgrgid_r` — get group database entry for a group ID34001 **SYNOPSIS**

```
34002 #include <grp.h>
34003
34003 struct group *getgrgid(gid_t gid);
34004 int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
34005               size_t bufsize, struct group **result);
```

34006 **DESCRIPTION**34007 The `getgrgid()` function shall search the group database for an entry with a matching *gid*.34008 The `getgrgid()` function need not be thread-safe.

34009 The `getgrgid_r()` function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *gid*. Storage referenced by the group structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to `sysconf(_SC_GETGR_R_SIZE_MAX)` returns either `-1` without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

34016 **RETURN VALUE**

34017 Upon successful completion, `getgrgid()` shall return a pointer to a **struct group** with the structure defined in `<grp.h>` with a matching entry if one is found. The `getgrgid()` function shall return a null pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be set to indicate the error.

34021 The return value may point to a static area which is overwritten by a subsequent call to `getgrent()`, `getgrgid()`, or `getgrnam()`.

34023 If successful, the `getgrgid_r()` function shall return zero; otherwise, an error number shall be returned to indicate the error.

34025 **ERRORS**34026 The `getgrgid()` and `getgrgid_r()` functions may fail if:

34027 [EIO] An I/O error has occurred.

34028 [EINTR] A signal was caught during `getgrgid()`.

34029 [EMFILE] All file descriptors available to the process are currently open.

34030 [ENFILE] The maximum allowable number of files is currently open in the system.

34031 The `getgrgid_r()` function may fail if:34032 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

34034 **EXAMPLES**

34035 Note that *sysconf*(*\_SC\_GETGR\_R\_SIZE\_MAX*) may return *-1* if there is no hard limit on the size  
 34036 of the buffer needed to store all the groups returned. This example shows how an application  
 34037 can allocate a buffer of sufficient size to work with *getgrgid\_r*().

```

34038 long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
34039 size_t len;
34040 if (initlen == -1)
34041     /* Default initial length. */
34042     len = 1024;
34043 else
34044     len = (size_t) initlen;
34045 struct group result;
34046 struct group *resultp;
34047 char *buffer = malloc(len);
34048 if (buffer == NULL)
34049     ...handle error...
34050 int e;
34051 while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
34052     {
34053     size_t newlen = 2 * len;
34054     if (newlen < len)
34055         ...handle error...
34056     len = newlen;
34057     char *newbuffer = realloc(buffer, len);
34058     if (newbuffer == NULL)
34059         ...handle error...
34060     buffer = newbuffer;
34061     }
34062 if (e != 0)
34063     ...handle error...
34064 free (buffer);

```

34065 **Finding an Entry in the Group Database**

34066 The following example uses *getgrgid*() to search the group database for a group ID that was  
 34067 previously stored in a *stat* structure, then prints out the group name if it is found. If the group is  
 34068 not found, the program prints the numeric value of the group for the entry.

```

34069 #include <sys/types.h>
34070 #include <grp.h>
34071 #include <stdio.h>
34072 ...
34073 struct stat statbuf;
34074 struct group *grp;
34075 ...
34076 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
34077     printf(" %-8.8s", grp->gr_name);
34078 else
34079     printf(" %-8d", statbuf.st_gid);
34080 ...

```

**getgrgid()**34081 **APPLICATION USAGE**

34082 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.  
 34083 If *errno* is set on return, an error occurred.

34084 The *getgrgid\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 34085 of possibly using a static data area that may be overwritten by each call.

34086 Portable applications should take into account that it is usual for an implementation to return -1  
 34087 from *sysconf()* indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

34088 **RATIONALE**

34089 None.

34090 **FUTURE DIRECTIONS**

34091 None.

34092 **SEE ALSO**

34093 *endgrent()*, *getgrnam()*, *sysconf()*

34094 XBD `<grp.h>`, `<sys/types.h>`

34095 **CHANGE HISTORY**

34096 First released in Issue 1. Derived from System V Release 2.0.

34097 **Issue 5**

34098 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 34099 VALUE section.

34100 The *getgrgid\_r()* function is included for alignment with the POSIX Threads Extension.

34101 A note indicating that the *getgrgid()* function need not be reentrant is added to the  
 34102 DESCRIPTION.

34103 **Issue 6**

34104 The *getgrgid\_r()* function is marked as part of the Thread-Safe Functions option.

34105 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 34106 describing matching the *gid*.

34107 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

34108 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

34109 The following new requirements on POSIX implementations derive from alignment with the  
 34110 Single UNIX Specification:

34111 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 34112 required for conforming implementations of previous POSIX specifications, it was not  
 34113 required for UNIX applications.

34114 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

34115 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

34116 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 34117 its avoidance of possibly using a static data area.

34118 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 34119 buffer from *bufsize* characters to bytes.

34120 **Issue 7**

- 34121 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 34122 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 34123 SD5-XSH-ERN-166 is applied.
- 34124 The *getgrgid\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 34125 A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc()*.
- 34126

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getgrnam()**34127 **NAME**34128 `getgrnam, getgrnam_r` — search group database for a name34129 **SYNOPSIS**

```
34130 #include <grp.h>
34131 struct group *getgrnam(const char *name);
34132 int getgrnam_r(const char *name, struct group *grp, char *buffer,
34133 size_t bufsize, struct group **result);
```

34134 **DESCRIPTION**34135 The `getgrnam()` function shall search the group database for an entry with a matching *name*.34136 The `getgrnam()` function need not be thread-safe.

34137 The `getgrnam_r()` function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *name*. Storage referenced by the **group** structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to `sysconf(_SC_GETGR_R_SIZE_MAX)` returns either `-1` without changing *errno* or an initial value suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result* on error or if the requested entry is not found.

34144 **RETURN VALUE**

34145 The `getgrnam()` function shall return a pointer to a **struct group** with the structure defined in `<grp.h>` with a matching entry if one is found. The `getgrnam()` function shall return a null pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be set to indicate the error.

34149 The return value may point to a static area which is overwritten by a subsequent call to `getgrent()`, `getgrgid()`, or `getgrnam()`.

34151 The `getgrnam_r()` function shall return zero on success or if the requested entry was not found and no error has occurred. If any error has occurred, an error number shall be returned to indicate the error.

34154 **ERRORS**34155 The `getgrnam()` and `getgrnam_r()` functions may fail if:

34156 [EIO] An I/O error has occurred.

34157 [EINTR] A signal was caught during `getgrnam()`.

34158 [EMFILE] All file descriptors available to the process are currently open.

34159 [ENFILE] The maximum allowable number of files is currently open in the system.

34160 The `getgrnam_r()` function may fail if:34161 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

34163 **EXAMPLES**

34164 Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size  
 34165 of the buffer needed to store all the groups returned. This example shows how an application  
 34166 can allocate a buffer of sufficient size to work with `getgrnam_r()`.

```

34167 long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
34168 size_t len;
34169 if (initlen == -1)
34170     /* Default initial length. */
34171     len = 1024;
34172 else
34173     len = (size_t) initlen;
34174 struct group result;
34175 struct group *resultp;
34176 char *buffer = malloc(len);
34177 if (buffer == NULL)
34178     ...handle error...
34179 int e;
34180 while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
34181        == ERANGE)
34182     {
34183     size_t newlen = 2 * len;
34184     if (newlen < len)
34185         ...handle error...
34186     len = newlen;
34187     char *newbuffer = realloc(buffer, len);
34188     if (newbuffer == NULL)
34189         ...handle error...
34190     buffer = newbuffer;
34191     }
34192 if (e != 0)
34193     ...handle error...
34194 free (buffer);

```

34195 **APPLICATION USAGE**

34196 Applications wishing to check for error situations should set `errno` to 0 before calling `getgrnam()`.  
 34197 If `errno` is set on return, an error occurred.

34198 The `getgrnam_r()` function is thread-safe and shall return values in a user-supplied buffer instead  
 34199 of possibly using a static data area that may be overwritten by each call.

34200 Portable applications should take into account that it is usual for an implementation to return `-1`  
 34201 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

34202 **RATIONALE**

34203 None.

34204 **FUTURE DIRECTIONS**

34205 None.

34206 **SEE ALSO**

34207 [\*endgrent\(\)\*](#), [\*getgrgid\(\)\*](#), [\*sysconf\(\)\*](#)

34208 XBD [\*<grp.h>\*](#), [\*<sys/types.h>\*](#)

**getgrnam()****34209 CHANGE HISTORY**

34210 First released in Issue 1. Derived from System V Release 2.0.

**34211 Issue 5**

34212 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
34213 VALUE section.

34214 The *getgrnam\_r()* function is included for alignment with the POSIX Threads Extension.

34215 A note indicating that the *getgrnam()* function need not be reentrant is added to the  
34216 DESCRIPTION.

**34217 Issue 6**

34218 The *getgrnam\_r()* function is marked as part of the Thread-Safe Functions option.

34219 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

34220 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

34221 The following new requirements on POSIX implementations derive from alignment with the  
34222 Single UNIX Specification:

- 34223 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
34224 required for conforming implementations of previous POSIX specifications, it was not  
34225 required for UNIX applications.
- 34226 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 34227 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

34228 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
34229 its avoidance of possibly using a static data area.

34230 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
34231 buffer from *bufsize* characters to bytes.

**34232 Issue 7**

34233 Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

34234 Austin Group Interpretation 1003.1-2001 #156 is applied.

34235 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

34236 SD5-XSH-ERN-166 is applied.

34237 The *getgrnam\_r()* function is moved from the Thread-Safe Functions option to the Base.

34238 A minor addition is made to the EXAMPLES section, reminding the application developer to  
34239 free memory allocated as if by *malloc()*.

**34240 NAME**

34241 `getgroups` — get supplementary group IDs

**34242 SYNOPSIS**

```
34243 #include <unistd.h>
34244 int getgroups(int gidsetsize, gid_t grouplist[]);
```

**34245 DESCRIPTION**

34246 The `getgroups()` function shall fill in the array `grouplist` with the current supplementary group  
 34247 IDs of the calling process. It is implementation-defined whether `getgroups()` also returns the  
 34248 effective group ID in the `grouplist` array.

34249 The `gidsetsize` argument specifies the number of elements in the array `grouplist`. The actual  
 34250 number of group IDs stored in the array shall be returned. The values of array entries with  
 34251 indices greater than or equal to the value returned are undefined.

34252 If `gidsetsize` is 0, `getgroups()` shall return the number of group IDs that it would otherwise return  
 34253 without modifying the array pointed to by `grouplist`.

34254 If the effective group ID of the process is returned with the supplementary group IDs, the value  
 34255 returned shall always be greater than or equal to one and less than or equal to the value of  
 34256 `{NGROUPS_MAX}+1`.

**34257 RETURN VALUE**

34258 Upon successful completion, the number of supplementary group IDs shall be returned. A  
 34259 return value of `-1` indicates failure and `errno` shall be set to indicate the error.

**34260 ERRORS**

34261 The `getgroups()` function shall fail if:

34262 [EINVAL] The `gidsetsize` argument is non-zero and less than the number of group IDs  
 34263 that would have been returned.

**34264 EXAMPLES****34265 Getting the Supplementary Group IDs of the Calling Process**

34266 The following example places the current supplementary group IDs of the calling process into  
 34267 the `group` array.

```
34268 #include <sys/types.h>
34269 #include <unistd.h>
34270 ...
34271 gid_t *group;
34272 int ngroups;
34273 long ngroups_max;
34274 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
34275 group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));
34276 ngroups = getgroups(ngroups_max, group);
```

**34277 APPLICATION USAGE**

34278 None.

**34279 RATIONALE**

34280 The related function `setgroups()` is a privileged operation and therefore is not covered by this  
 34281 volume of POSIX.1-2008.

34282 As implied by the definition of supplementary groups, the effective group ID may appear in the

**getgroups()**

34283 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but  
 34284 the application needs to call *getegid()* to be sure of getting all of the information. Various  
 34285 implementation variations and administrative sequences cause the set of groups appearing in  
 34286 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,  
 34287 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a  
 34288 process and its parents could affect the details of the result.)

34289 Application developers should note that {NGROUPS\_MAX} is not necessarily a constant on all  
 34290 implementations.

**FUTURE DIRECTIONS**

34291 None.

**SEE ALSO**

34292 *getegid()*, *setgid()*

34293 XBD [<sys/types.h>](#), [<unistd.h>](#)

**CHANGE HISTORY**

34294 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**Issue 5**

34295 Normative text previously in the APPLICATION USAGE section is moved to the  
 34296 DESCRIPTION.

**Issue 6**

34297 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

34298 The following new requirements on POSIX implementations derive from alignment with the  
 34299 Single UNIX Specification:

- 34300 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 34301 required for conforming implementations of previous POSIX specifications, it was not  
 34302 required for UNIX applications.
- 34303 • A return value of 0 is not permitted, because {NGROUPS\_MAX} cannot be 0. This is a FIPS  
 34304 requirement.

34305 The following changes were made to align with the IEEE P1003.1a draft standard:

- 34306 • An explanation is added that the effective group ID may be included in the supplementary  
 34307 group list.

*System Interfaces***gethostent()**

34313 **NAME**  
34314       gethostent — network host database functions

34315 **SYNOPSIS**  
34316       #include <netdb.h>  
34317       struct hostent \*gethostent(void);

34318 **DESCRIPTION**  
34319       Refer to *endhostent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**gethostid()**34320 **NAME**

34321           gethostid — get an identifier for the current host

34322 **SYNOPSIS**34323 XSI       #include <unistd.h>  
34324       long gethostid(void);34325 **DESCRIPTION**34326       The *gethostid()* function shall retrieve a 32-bit identifier for the current host.34327 **RETURN VALUE**34328       Upon successful completion, *gethostid()* shall return an identifier for the current host.34329 **ERRORS**

34330       No errors are defined.

34331 **EXAMPLES**

34332       None.

34333 **APPLICATION USAGE**

34334       This volume of POSIX.1-2008 does not define the domain in which the return value is unique.

34335 **RATIONALE**

34336       None.

34337 **FUTURE DIRECTIONS**

34338       None.

34339 **SEE ALSO**34340       *initstate()*

34341       XBD &lt;unistd.h&gt;

34342 **CHANGE HISTORY**

34343       First released in Issue 4, Version 2.

34344 **Issue 5**

34345       Moved from X/OPEN UNIX extension to BASE.

34346 **NAME**

34347       gethostname — get name of current host

34348 **SYNOPSIS**

34349       #include &lt;unistd.h&gt;

34350       int gethostname(char \*name, size\_t namelen);

34351 **DESCRIPTION**

34352       The *gethostname()* function shall return the standard host name for the current machine. The

34353       *namelen* argument shall specify the size of the array pointed to by the *name* argument. The

34354       returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold

34355       the host name, then the returned name shall be truncated and it is unspecified whether the

34356       returned name is null-terminated.

34357       Host names are limited to {HOST\_NAME\_MAX} bytes.

34358 **RETURN VALUE**

34359       Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

34360 **ERRORS**

34361       No errors are defined.

34362 **EXAMPLES**

34363       None.

34364 **APPLICATION USAGE**

34365       None.

34366 **RATIONALE**

34367       None.

34368 **FUTURE DIRECTIONS**

34369       None.

34370 **SEE ALSO**34371       *gethostid()*, *uname()*

34372       XBD &lt;unistd.h&gt;

34373 **CHANGE HISTORY**

34374       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

34375       The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from

34376       *socklen\_t* to *size\_t*.

**getitimer()**34377 **NAME**34378 `getitimer, setitimer` — get and set value of interval timer34379 **SYNOPSIS**

```

34380 OB XSI #include <sys/time.h>
34381
34381 int getitimer(int which, struct itimerval *value);
34382 int setitimer(int which, const struct itimerval *restrict value,
34383              struct itimerval *restrict ovalue);

```

34384 **DESCRIPTION**

34385 The `getitimer()` function shall store the current value of the timer specified by *which* into the  
 34386 structure pointed to by *value*. The `setitimer()` function shall set the timer specified by *which* to the  
 34387 value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store the  
 34388 previous value of the timer in the structure pointed to by *ovalue*.

34389 A timer value is defined by the **itimerval** structure, specified in `<sys/time.h>`. If *it\_value* is non-  
 34390 zero, it shall indicate the time to the next timer expiration. If *it\_interval* is non-zero, it shall  
 34391 specify a value to be used in reloading *it\_value* when the timer expires. Setting *it\_value* to 0 shall  
 34392 disable a timer, regardless of the value of *it\_interval*. Setting *it\_interval* to 0 shall disable a timer  
 34393 after its next expiration (assuming *it\_value* is non-zero).

34394 Implementations may place limitations on the granularity of timer values. For each interval  
 34395 timer, if the requested timer value requires a finer granularity than the implementation supports,  
 34396 the actual timer value shall be rounded up to the next supported value.

34397 An XSI-conforming implementation provides each process with at least three interval timers,  
 34398 which are indicated by the *which* argument:

34399 **ITIMER\_PROF** Decrements both in process virtual time and when the system is running  
 34400 on behalf of the process. It is designed to be used by interpreters in  
 34401 statistically profiling the execution of interpreted programs. Each time the  
 34402 **ITIMER\_PROF** timer expires, the **SIGPROF** signal is delivered.

34403 **ITIMER\_REAL** Decrements in real time. A **SIGALRM** signal is delivered when this timer  
 34404 expires.

34405 **ITIMER\_VIRTUAL** Decrements in process virtual time. It runs only when the process is  
 34406 executing. A **SIGVTALRM** signal is delivered when it expires.

34407 The interaction between `setitimer()` and `alarm()` or `sleep()` is unspecified.

34408 **RETURN VALUE**

34409 Upon successful completion, `getitimer()` or `setitimer()` shall return 0; otherwise, -1 shall be  
 34410 returned and *errno* set to indicate the error.

34411 **ERRORS**

34412 The `setitimer()` function shall fail if:

34413 **[EINVAL]** The *value* argument is not in canonical form. (In canonical form, the number of  
 34414 microseconds is a non-negative integer less than 1 000 000 and the number of  
 34415 seconds is a non-negative integer.)

34416 The `getitimer()` and `setitimer()` functions may fail if:

34417 **[EINVAL]** The *which* argument is not recognized.

34418 **EXAMPLES**

34419 None.

34420 **APPLICATION USAGE**34421 Applications should use the *timer\_gettime()* and *timer\_settime()* functions instead of the  
34422 obsolescent *getitimer()* and *setitimer()* functions, respectively.34423 **RATIONALE**

34424 None.

34425 **FUTURE DIRECTIONS**34426 The *getitimer()* and *setitimer()* functions may be removed in a future version.34427 **SEE ALSO**34428 *alarm()*, *exec*, *sleep()*, *timer\_getoverrun()*34429 XBD [<signal.h>](#), [<sys/time.h>](#)34430 **CHANGE HISTORY**

34431 First released in Issue 4, Version 2.

34432 **Issue 5**

34433 Moved from X/OPEN UNIX extension to BASE.

34434 **Issue 6**34435 The **restrict** keyword is added to the *setitimer()* prototype for alignment with the  
34436 ISO/IEC 9899:1999 standard.34437 **Issue 7**34438 The *getitimer()* and *setitimer()* functions are marked obsolescent.

**getline()**34439 **NAME**34440 `getline` — read a delimited record from *stream*34441 **SYNOPSIS**

```
34442 CX #include <stdio.h>  
34443      ssize_t getline(char **restrict lineptr, size_t *restrict n,  
34444                  FILE *restrict stream);
```

34445 **DESCRIPTION**34446 Refer to *getdelim()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

34447 **NAME**34448 `getlogin, getlogin_r` — get login name34449 **SYNOPSIS**

```
34450 #include <unistd.h>
34451 char *getlogin(void);
34452 int getlogin_r(char *name, size_t namesize);
```

34453 **DESCRIPTION**

34454 The `getlogin()` function shall return a pointer to a string containing the user name associated by  
 34455 the login activity with the controlling terminal of the current process. If `getlogin()` returns a non-  
 34456 null pointer, then that pointer points to the name that the user logged in under, even if there are  
 34457 several login names with the same user ID.

34458 The `getlogin()` function need not be thread-safe.

34459 The `getlogin_r()` function shall put the name associated by the login activity with the controlling  
 34460 terminal of the current process in the character array pointed to by `name`. The array is `namesize`  
 34461 characters long and should have space for the name and the terminating null character. The  
 34462 maximum size of the login name is {LOGIN\_NAME\_MAX}.

34463 If `getlogin_r()` is successful, `name` points to the name the user used at login, even if there are  
 34464 several login names with the same user ID.

34465 **RETURN VALUE**

34466 Upon successful completion, `getlogin()` shall return a pointer to the login name or a null pointer  
 34467 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set `errno` to  
 34468 indicate the error.

34469 The return value from `getlogin()` may point to static data whose content is overwritten by each  
 34470 call.

34471 If successful, the `getlogin_r()` function shall return zero; otherwise, an error number shall be  
 34472 returned to indicate the error.

34473 **ERRORS**

34474 These functions may fail if:

- 34475 [EMFILE] All file descriptors available to the process are currently open.
- 34476 [ENFILE] The maximum allowable number of files is currently open in the system.
- 34477 [ENXIO] The calling process has no controlling terminal.

34478 The `getlogin_r()` function may fail if:

- 34479 [ERANGE] The value of `namesize` is smaller than the length of the string to be returned  
 34480 including the terminating null character.

34481 **EXAMPLES**34482 **Getting the User Login Name**

34483 The following example calls the `getlogin()` function to obtain the name of the user associated  
 34484 with the calling process, and passes this information to the `getpwnam()` function to get the  
 34485 associated user database information.

```
34486 #include <unistd.h>
34487 #include <sys/types.h>
34488 #include <pwd.h>
```

**getlogin()**

```

34489     #include <stdio.h>
34490     ...
34491     char *lgn;
34492     struct passwd *pw;
34493     ...
34494     if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
34495         fprintf(stderr, "Get of user information failed.\n"); exit(1);
34496     }

```

**APPLICATION USAGE**

Three names associated with the current process can be determined: *getpwuid(geteuid())* shall return the name associated with the effective user ID of the process; *getlogin()* shall return the name associated with the current login activity; and *getpwuid(getuid())* shall return the name associated with the real user ID of the process.

The *getlogin\_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

**RATIONALE**

The *getlogin()* function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()* function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the *cuserid()* function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(geteuid())
```

while the information provided by historical implementations of *cuserid()* can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*

XBD [<limits.h>](#), [<unistd.h>](#)

**CHANGE HISTORY**

First released in Issue 1. Derived from System V Release 2.0.

**Issue 5**

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getlogin\_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getlogin()* function need not be reentrant is added to the DESCRIPTION.

34531 **Issue 6**

34532 The *getlogin\_r()* function is marked as part of the Thread-Safe Functions option.

34533 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

34534 The following new requirements on POSIX implementations derive from alignment with the  
34535 Single UNIX Specification:

34536 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

34537 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

34538 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
34539 its avoidance of possibly using a static data area.

34540 **Issue 7**

34541 Austin Group Interpretation 1003.1-2001 #156 is applied.

34542 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

34543 The *getlogin\_r()* function is moved from the Thread-Safe Functions option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getmsg()**34544 **NAME**34545 `getmsg, getpmsg` — receive next message from a STREAMS file (**STREAMS**)34546 **SYNOPSIS**

```

34547 OB XSR #include <stropts.h>
34548 int getmsg(int fildev, struct strbuf *restrict ctlptr,
34549           struct strbuf *restrict dataptr, int *restrict flagsp);
34550 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
34551            struct strbuf *restrict dataptr, int *restrict bandp,
34552            int *restrict flagsp);

```

34553 **DESCRIPTION**

34554 The `getmsg()` function shall retrieve the contents of a message located at the head of the  
 34555 STREAM head read queue associated with a STREAMS file and place the contents into one or  
 34556 more buffers. The message contains either a data part, a control part, or both. The data and  
 34557 control parts of the message shall be placed into separate buffers, as described below. The  
 34558 semantics of each part are defined by the originator of the message.

34559 The `getpmsg()` function shall be equivalent to `getmsg()`, except that it provides finer control over  
 34560 the priority of the messages received. Except where noted, all requirements on `getmsg()` also  
 34561 pertain to `getpmsg()`.

34562 The *fildev* argument specifies a file descriptor referencing a STREAMS-based file.

34563 The *ctlptr* and *dataptr* arguments each point to a `strbuf` structure, in which the *buf* member points  
 34564 to a buffer in which the data or control information is to be placed, and the *maxlen* member  
 34565 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall  
 34566 contain the number of bytes of data or control information actually received. The *len* member  
 34567 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or  
 34568 control information is present in the message.

34569 When `getmsg()` is called, *flagsp* should point to an integer that indicates the type of message the  
 34570 process is able to receive. This is described further below.

34571 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold  
 34572 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the  
 34573 control (or data) part of the message shall not be processed and shall be left on the STREAM  
 34574 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the  
 34575 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part  
 34576 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0  
 34577 and there are more than 0 bytes of control (or data) information, that information shall be left on  
 34578 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the  
 34579 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder  
 34580 of the message shall be left on the STREAM head read queue and a non-zero return value shall  
 34581 be provided.

34582 By default, `getmsg()` shall process the first available message on the STREAM head read queue.  
 34583 However, a process may choose to retrieve only high-priority messages by setting the integer  
 34584 pointed to by *flagsp* to `RS_HIPRI`. In this case, `getmsg()` shall only process the next message if it is  
 34585 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall  
 34586 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to `RS_HIPRI` if a  
 34587 high-priority message was retrieved, or 0 otherwise.

34588 For `getpmsg()`, the flags are different. The *flagsp* argument points to a bitmask with the following  
 34589 mutually-exclusive flags defined: `MSG_HIPRI`, `MSG_BAND`, and `MSG_ANY`. Like `getmsg()`,

34590 *getpmsg()* shall process the first available message on the STREAM head read queue. A process  
 34591 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to  
 34592 MSG\_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process  
 34593 the next message if it is a high-priority message. In a similar manner, a process may choose to  
 34594 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to  
 34595 MSG\_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,  
 34596 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than  
 34597 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the  
 34598 first message off the queue, the integer pointed to by *flagsp* should be set to MSG\_ANY and the  
 34599 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-  
 34600 priority message, the integer pointed to by *flagsp* shall be set to MSG\_HIPRI and the integer  
 34601 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to  
 34602 MSG\_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

34603 If O\_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type  
 34604 specified by *flagsp* is available at the front of the STREAM head read queue. If O\_NONBLOCK is  
 34605 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and  
 34606 *getpmsg()* shall fail and set *errno* to [EAGAIN].

34607 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*  
 34608 shall continue to operate normally, as described above, until the STREAM head read queue is  
 34609 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

#### 34610 RETURN VALUE

34611 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value  
 34612 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates  
 34613 that more control information is waiting for retrieval. A return value of MOREDATA indicates  
 34614 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL  
 34615 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and  
 34616 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher  
 34617 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*  
 34618 shall retrieve that higher-priority message before retrieving the remainder of the previous  
 34619 message.

34620 If the high priority control part of the message is consumed, the message shall be placed back on  
 34621 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve  
 34622 the remainder of the message. If, however, a priority message arrives or already exists on the  
 34623 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority  
 34624 message before retrieving the remainder of the message that was put back.

34625 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

#### 34626 ERRORS

34627 The *getmsg()* and *getpmsg()* functions shall fail if:

34628	[EAGAIN]	The O_NONBLOCK flag is set and no messages are available.
34629	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor open for reading.
34630	[EBADMSG]	The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a pending file descriptor is at the STREAM head.
34631		
34632	[EINTR]	A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .
34633	[EINVAL]	An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
34634		
34635		

**getmsg()**

34636 [ENOSTR] A STREAM is not associated with *fildev*.

34637 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an

34638 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of

34639 *getmsg()* or *getpmsg()* but reflects the prior error.

34640 **EXAMPLES**34641 **Getting Any Message**

34642 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call

34643 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,

34644 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,

34645 respectively.

```
34646 #include <stropts.h>
34647 ...
34648 int fd;
34649 char ctrlbuf[128];
34650 char databuf[512];
34651 struct strbuf ctrl;
34652 struct strbuf data;
34653 int flags = 0;
34654 int ret;

34655 ctrl.buf = ctrlbuf;
34656 ctrl.maxlen = sizeof(ctrlbuf);

34657 data.buf = databuf;
34658 data.maxlen = sizeof(databuf);

34659 ret = getmsg (fd, &ctrl, &data, &flags);
```

34660 **Getting the First Message off the Queue**

34661 In the following example, the call to *getpmsg()* retrieves the first available message on the

34662 associated STREAM-head read queue.

```
34663 #include <stropts.h>
34664 ...
34665 int fd;
34666 char ctrlbuf[128];
34667 char databuf[512];
34668 struct strbuf ctrl;
34669 struct strbuf data;
34670 int band = 0;
34671 int flags = MSG_ANY;
34672 int ret;

34673 ctrl.buf = ctrlbuf;
34674 ctrl.maxlen = sizeof(ctrlbuf);

34675 data.buf = databuf;
34676 data.maxlen = sizeof(databuf);

34677 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

**34678 APPLICATION USAGE**

34679 None.

**34680 RATIONALE**

34681 None.

**34682 FUTURE DIRECTIONS**

34683 The *getmsg()* and *getpmsg()* functions may be removed in a future version.

**34684 SEE ALSO**

34685 [Section 2.6](#) (on page 494), *poll()*, *putmsg()*, *read()*, *write()*

34686 XBD [<stropts.h>](#)

**34687 CHANGE HISTORY**

34688 First released in Issue 4, Version 2.

**34689 Issue 5**

34690 Moved from X/OPEN UNIX extension to BASE.

34691 A paragraph regarding “high-priority control parts of messages” is added to the RETURN  
34692 VALUE section.

**34693 Issue 6**

34694 This function is marked as part of the XSI STREAMS Option Group.

34695 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the  
34696 ISO/IEC 9899:1999 standard.

**34697 Issue 7**

34698 The *getmsg()* and *getpmsg()* functions are marked obsolescent.

**getnameinfo()**34699 **NAME**34700 `getnameinfo` — get name information34701 **SYNOPSIS**34702 `#include <sys/socket.h>`34703 `#include <netdb.h>`

```
34704 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
34705                char *restrict node, socklen_t nodelen, char *restrict service,
34706                socklen_t servicelen, int flags);
```

34707 **DESCRIPTION**

34708 The `getnameinfo()` function shall translate a socket address to a node name and service location,  
 34709 all of which are defined as in `freeaddrinfo()`.

34710 The `sa` argument points to a socket address structure to be translated.

34711 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible  
 34712 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node  
 34713 name for that IPv4 address.

34714 If the address is the IPv6 unspecified address ("`:::`"), a lookup shall not be performed and the  
 34715 behavior shall be the same as when the node's name cannot be located.

34716 If the `node` argument is non-NULL and the `nodelen` argument is non-zero, then the `node` argument  
 34717 points to a buffer able to contain up to `nodelen` characters that receives the node name as a null-  
 34718 terminated string. If the `node` argument is NULL or the `nodelen` argument is zero, the node name  
 34719 shall not be returned. If the node's name cannot be located, the numeric form of the address  
 34720 contained in the socket address structure pointed to by the `sa` argument is returned instead of its  
 34721 name.

34722 If the `service` argument is non-NULL and the `servicelen` argument is non-zero, then the `service`  
 34723 argument points to a buffer able to contain up to `servicelen` bytes that receives the service name  
 34724 as a null-terminated string. If the `service` argument is NULL or the `servicelen` argument is zero,  
 34725 the service name shall not be returned. If the service's name cannot be located, the numeric form  
 34726 of the service address (for example, its port number) shall be returned instead of its name.

34727 The `flags` argument is a flag that changes the default actions of the function. By default the fully-  
 34728 qualified domain name (FQDN) for the host shall be returned, but:

- 34729 • If the flag bit `NI_NOFQDN` is set, only the node name portion of the FQDN shall be  
 34730 returned for local hosts.
- 34731 • If the flag bit `NI_NUMERICHOST` is set, the numeric form of the address contained in the  
 34732 socket address structure pointed to by the `sa` argument shall be returned instead of its  
 34733 name.
- 34734 • If the flag bit `NI_NAMEREQD` is set, an error shall be returned if the host's name cannot  
 34735 be located.
- 34736 • If the flag bit `NI_NUMERICSERV` is set, the numeric form of the service address shall be  
 34737 returned (for example, its port number) instead of its name.
- 34738 • If the flag bit `NI_NUMERICSERVICE` is set, the numeric form of the scope identifier shall be  
 34739 returned (for example, interface index) instead of its name. This flag shall be ignored if the  
 34740 `sa` argument is not an IPv6 address.
- 34741 • If the flag bit `NI_DGRAM` is set, this indicates that the service is a datagram service  
 34742 (`SOCK_DGRAM`). The default behavior shall assume that the service is a stream service  
 34743 (`SOCK_STREAM`).

34744 **Notes:**

- 34745 1. The two NI\_NUMERICxxx flags are required to support the -n flag that many  
34746 commands provide.
- 34747 2. The NI\_DGRAM flag is required for the few AF\_INET and AF\_INET6 port numbers (for  
34748 example, [512,514]) that represent different services for UDP and TCP.

34749 The *getnameinfo()* function shall be thread-safe.

34750 **RETURN VALUE**

34751 A zero return value for *getnameinfo()* indicates successful completion; a non-zero return value  
34752 indicates failure. The possible values for the failures are listed in the ERRORS section.

34753 Upon successful completion, *getnameinfo()* shall return the *node* and *service* names, if requested,  
34754 in the buffers provided. The returned names are always null-terminated strings.

34755 **ERRORS**

34756 The *getnameinfo()* function shall fail and return the corresponding value if:

34757 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

34758 [EAI\_BADFLAGS]

34759 The *flags* had an invalid value.

34760 [EAI\_FAIL] A non-recoverable error occurred.

34761 [EAI\_FAMILY] The address family was not recognized or the address length was invalid for  
34762 the specified family.

34763 [EAI\_MEMORY] There was a memory allocation failure.

34764 [EAI\_NONAME] The name does not resolve for the supplied parameters.

34765 NI\_NAMEREQD is set and the host's name cannot be located, or both  
34766 *nodename* and *servname* were null.

34767 [EAI\_OVERFLOW]

34768 An argument buffer overflowed. The buffer pointed to by the *node* argument  
34769 or the *service* argument was too small.

34770 [EAI\_SYSTEM] A system error occurred. The error code can be found in *errno*.

34771 **EXAMPLES**

34772 None.

34773 **APPLICATION USAGE**

34774 If the returned values are to be used as part of any further name resolution (for example, passed  
34775 to *getaddrinfo()*), applications should provide buffers large enough to store any result possible on  
34776 the system.

34777 Given the IPv4-mapped IPv6 address "::ffff:1.2.3.4", the implementation performs a  
34778 lookup as if the socket address structure contains the IPv4 address "1.2.3.4".

34779 The IPv6 unspecified address ("::") and the IPv6 loopback address ("::1") are not  
34780 IPv4-compatible addresses.

34781 **RATIONALE**

34782 None.

**getnameinfo()**34783 **FUTURE DIRECTIONS**

34784 None.

34785 **SEE ALSO**34786 *endservent()*, *freeaddrinfo()*, *gai\_strerror()*, *inet\_ntop()*, *socket()*34787 XBD [<netdb.h>](#), [<sys/socket.h>](#)34788 **CHANGE HISTORY**

34789 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

34790 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the  
34791 ISO/IEC 9899:1999 standard.34792 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in  
34793 the SYNOPSIS and DESCRIPTION for alignment with IPv6.34794 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the  
34795 [EAL\_OVERFLOW] error to the ERRORS section.34796 **Issue 7**34797 SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified  
34798 address.

34799 **NAME**

34800 getnetbyaddr, getnetbyname, getnetent — network database functions

34801 **SYNOPSIS**

34802 #include &lt;netdb.h&gt;

34803 struct netent \*getnetbyaddr(uint32\_t net, int type);

34804 struct netent \*getnetbyname(const char \*name);

34805 struct netent \*getnetent(void);

34806 **DESCRIPTION**34807 Refer to *endnetent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getopt()**34808 **NAME**34809 `getopt, optarg, opterr, optind, optopt` — command option parsing34810 **SYNOPSIS**

```
34811 #include <unistd.h>
34812 int getopt(int argc, char * const argv[], const char *optstring);
34813 extern char *optarg;
34814 extern int opterr, optind, optopt;
```

34815 **DESCRIPTION**

34816 The `getopt()` function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,  
34817 6, 7, 9, and 10 in XBD Section 12.2 (on page 215).

34818 The parameters `argc` and `argv` are the argument count and argument array as passed to `main()`  
34819 (see `exec()`). The argument `optstring` is a string of recognized option characters; if a character is  
34820 followed by a <colon>, the option takes an argument. All option characters allowed by Utility  
34821 Syntax Guideline 3 are allowed in `optstring`. The implementation may accept other characters as  
34822 an extension.

34823 The variable `optind` is the index of the next element of the `argv[]` vector to be processed. It shall  
34824 be initialized to 1 by the system, and `getopt()` shall update it when it finishes with each element  
34825 of `argv[]`. When an element of `argv[]` contains multiple option characters, it is unspecified how  
34826 `getopt()` determines which options have already been processed.

34827 The `getopt()` function shall return the next option character (if one is found) from `argv` that  
34828 matches a character in `optstring`, if there is one that matches. If the option takes an argument,  
34829 `getopt()` shall set the variable `optarg` to point to the option-argument as follows:

- 34830 1. If the option was the last character in the string pointed to by an element of `argv`, then  
34831 `optarg` shall contain the next element of `argv`, and `optind` shall be incremented by 2. If the  
34832 resulting value of `optind` is greater than `argc`, this indicates a missing option-argument,  
34833 and `getopt()` shall return an error indication.
- 34834 2. Otherwise, `optarg` shall point to the string following the option character in that element  
34835 of `argv`, and `optind` shall be incremented by 1.

34836 If, when `getopt()` is called:

```
34837 argv[optind] is a null pointer
34838 *argv[optind] is not the character -
34839 argv[optind] points to the string "-"
```

34840 `getopt()` shall return `-1` without changing `optind`. If:

```
34841 argv[optind] points to the string "--"
```

34842 `getopt()` shall return `-1` after incrementing `optind`.

34843 If `getopt()` encounters an option character that is not contained in `optstring`, it shall return the  
34844 <question-mark> ('?') character. If it detects a missing option-argument, it shall return the  
34845 <colon> character (':') if the first character of `optstring` was a <colon>, or a <question-mark>  
34846 character ('?') otherwise. In either case, `getopt()` shall set the variable `optopt` to the option  
34847 character that caused the error. If the application has not set the variable `opterr` to 0 and the first  
34848 character of `optstring` is not a <colon>, `getopt()` shall also print a diagnostic message to `stderr`  
34849 in the format specified for the `getopts` utility.

34850 The `getopt()` function need not be thread-safe.

34851 **RETURN VALUE**

34852 The *getopt()* function shall return the next option character specified on the command line.

34853 A <colon> (':') shall be returned if *getopt()* detects a missing argument and the first character  
34854 of *optstring* was a <colon> (':').

34855 A <question-mark> ('?') shall be returned if *getopt()* encounters an option character not in  
34856 *optstring* or detects a missing argument and the first character of *optstring* was not a <colon>  
34857 (':').

34858 Otherwise, *getopt()* shall return -1 when all command line options are parsed.

34859 **ERRORS**

34860 No errors are defined.

34861 **EXAMPLES**34862 **Parsing Command Line Options**

34863 The following code fragment shows how you might process the arguments for a utility that can  
34864 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require  
34865 arguments:

```
34866 #include <unistd.h>
34867
34868 int
34869 main(int argc, char *argv[ ])
34870 {
34871     int c;
34872     int bflg, aflag, errflag;
34873     char *ifile;
34874     char *ofile;
34875     extern char *optarg;
34876     extern int optind, optopt;
34877     . . .
34878     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
34879         switch(c)
34880         case 'a':
34881             if (bflg)
34882                 errflag++;
34883             else
34884                 aflag++;
34885             break;
34886         case 'b':
34887             if (aflag)
34888                 errflag++;
34889             else {
34890                 bflg++;
34891                 bproc();
34892             }
34893             break;
34894         case 'f':
34895             ifile = optarg;
34896             break;
34897         case 'o':
34898             ofile = optarg;
```

**getopt()**

```

34898         break;
34899         case ':': /* -f or -o without operand */
34900             fprintf(stderr,
34901                 "Option -%c requires an operand\n", optopt);
34902             errflg++;
34903             break;
34904         case '?':
34905             fprintf(stderr,
34906                 "Unrecognized option: -%c\n", optopt);
34907             errflg++;
34908     }
34909 }
34910 if (errflg) {
34911     fprintf(stderr, "usage: . . . ");
34912     exit(2);
34913 }
34914 for ( ; optind < argc; optind++) {
34915     if (access(argv[optind], R_OK)) {
34916         . . .
34917     }

```

34918 This code accepts any of the following as equivalent:

```

34919 cmd -ao arg path path
34920 cmd -a -o arg path path
34921 cmd -o arg -a path path
34922 cmd -a -o arg -- path path
34923 cmd -a -oarg path path
34924 cmd -aoarg path path

```

### 34925 **Checking Options and Arguments**

34926 The following example parses a set of command line options and prints messages to standard  
34927 output for each option and argument that it encounters.

```

34928 #include <unistd.h>
34929 #include <stdio.h>
34930 ...
34931 int c;
34932 char *filename;
34933 extern char *optarg;
34934 extern int optind, optopt, opterr;
34935 . . .
34936 while ((c = getopt(argc, argv, "abf:")) != -1) {
34937     switch(c) {
34938         case 'a':
34939             printf("a is set\n");
34940             break;
34941         case 'b':
34942             printf("b is set\n");
34943             break;
34944         case 'f':
34945             filename = optarg;

```

```

34946         printf("filename is %s\n", filename);
34947         break;
34948     case ':':
34949         printf("-%c without filename\n", optopt);
34950         break;
34951     case '?':
34952         printf("unknown arg %c\n", optopt);
34953         break;
34954     }
34955 }

```

### 34956 Selecting Options from the Command Line

34957 The following example selects the type of database routines the user wants to use based on the  
34958 *Options* argument.

```

34959 #include <unistd.h>
34960 #include <string.h>
34961 ...
34962 char *Options = "hdbt1";
34963 ...
34964 int dbtype, i;
34965 char c;
34966 char *st;
34967 ...
34968 dbtype = 0;
34969 while ((c = getopt(argc, argv, Options)) != -1) {
34970     if ((st = strchr(Options, c)) != NULL) {
34971         dbtype = st - Options;
34972         break;
34973     }
34974 }

```

### 34975 APPLICATION USAGE

34976 The *getopt()* function is only required to support option characters included in Utility Syntax  
34977 Guideline 3. Many historical implementations of *getopt()* support other characters as options.  
34978 This is an allowed extension, but applications that use extensions are not maximally portable.  
34979 Note that support for multi-byte option characters is only possible when such characters can be  
34980 represented as type **int**.

### 34981 RATIONALE

34982 The *optopt* variable represents historical practice and allows the application to obtain the identity  
34983 of the invalid option.

34984 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with  
34985 option-arguments whether separated from the option by <blank> characters or not. Note that  
34986 the requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

34987 The *getopt()* function shall return  $-1$ , rather than EOF, so that **<stdio.h>** is not required.

34988 The special significance of a <colon> as the first character of *optstring* makes *getopt()* consistent  
34989 with the *getopts* utility. It allows an application to make a distinction between a missing  
34990 argument and an incorrect option letter without having to examine the option letter. It is true  
34991 that a missing argument can only be detected in one case, but that is a case that has to be  
34992 considered.

**getopt()**34993 **FUTURE DIRECTIONS**

34994 None.

34995 **SEE ALSO**34996 *exec*34997 XBD [Section 12.2](#) (on page 215), [<unistd.h>](#)34998 **CHANGE HISTORY**

34999 First released in Issue 1. Derived from Issue 1 of the SVID.

35000 **Issue 5**35001 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.35002 **Issue 6**

35003 IEEE PASC Interpretation 1003.2 #150 is applied.

35004 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35005 **NAME**

35006 getpeername — get the name of the peer socket

35007 **SYNOPSIS**

35008 #include &lt;sys/socket.h&gt;

35009 int getpeername(int *socket*, struct sockaddr \*restrict *address*,  
35010 socklen\_t \*restrict *address\_len*);35011 **DESCRIPTION**35012 The *getpeername()* function shall retrieve the peer address of the specified socket, store this  
35013 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this  
35014 address in the object pointed to by the *address\_len* argument.35015 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
35016 the stored address shall be truncated.35017 If the protocol permits connections by unbound clients, and the peer is not bound, then the  
35018 value stored in the object pointed to by *address* is unspecified.35019 **RETURN VALUE**35020 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
35021 indicate the error.35022 **ERRORS**35023 The *getpeername()* function shall fail if:35024 [EBADF] The *socket* argument is not a valid file descriptor.

35025 [EINVAL] The socket has been shut down.

35026 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

35027 [ENOTSOCK] The *socket* argument does not refer to a socket.

35028 [EOPNOTSUPP] The operation is not supported for the socket protocol.

35029 The *getpeername()* function may fail if:

35030 [ENOBUFS] Insufficient resources were available in the system to complete the call.

35031 **EXAMPLES**

35032 None.

35033 **APPLICATION USAGE**

35034 None.

35035 **RATIONALE**

35036 None.

35037 **FUTURE DIRECTIONS**

35038 None.

35039 **SEE ALSO**35040 *accept()*, *bind()*, *getsockname()*, *socket()*

35041 XBD &lt;sys/socket.h&gt;

**getpeername()***System Interfaces*35042 **CHANGE HISTORY**

35043 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35044 The **restrict** keyword is added to the *getpeername()* prototype for alignment with the  
35045 ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35046 **NAME**

35047 getpgid — get the process group ID for a process

35048 **SYNOPSIS**

35049 #include &lt;unistd.h&gt;

35050 pid\_t getpgid(pid\_t pid);

35051 **DESCRIPTION**35052 The *getpgid()* function shall return the process group ID of the process whose process ID is equal  
35053 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.35054 **RETURN VALUE**35055 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return  
35056 (**pid\_t**)-1 and set *errno* to indicate the error.35057 **ERRORS**35058 The *getpgid()* function shall fail if:35059 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the  
35060 calling process, and the implementation does not allow access to the process  
35061 group ID of that process from the calling process.35062 [ESRCH] There is no process with a process ID equal to *pid*.35063 The *getpgid()* function may fail if:35064 [EINVAL] The value of the *pid* argument is invalid.35065 **EXAMPLES**

35066 None.

35067 **APPLICATION USAGE**

35068 None.

35069 **RATIONALE**

35070 None.

35071 **FUTURE DIRECTIONS**

35072 None.

35073 **SEE ALSO**35074 *exec*, *fork()*, *getpgid()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*

35075 XBD &lt;unistd.h&gt;

35076 **CHANGE HISTORY**

35077 First released in Issue 4, Version 2.

35078 **Issue 5**

35079 Moved from X/OPEN UNIX extension to BASE.

35080 **Issue 7**35081 The *getpgid()* function is moved from the XSI option to the Base.

**getpgrp()**35082 **NAME**35083 `getpgrp` — get the process group ID of the calling process35084 **SYNOPSIS**35085 `#include <unistd.h>`  
35086 `pid_t getpgrp(void);`35087 **DESCRIPTION**35088 The `getpgrp()` function shall return the process group ID of the calling process.35089 **RETURN VALUE**35090 The `getpgrp()` function shall always be successful and no return value is reserved to indicate an  
35091 error.35092 **ERRORS**

35093 No errors are defined.

35094 **EXAMPLES**

35095 None.

35096 **APPLICATION USAGE**

35097 None.

35098 **RATIONALE**35099 4.3 BSD provides a `getpgrp()` function that returns the process group ID for a specified process.  
35100 Although this function supports job control, all known job control shells always specify the  
35101 calling process with this function. Thus, the simpler System V `getpgrp()` suffices, and the added  
35102 complexity of the 4.3 BSD `getpgrp()` is provided by the XSI extension `getpgid()`.35103 **FUTURE DIRECTIONS**

35104 None.

35105 **SEE ALSO**35106 `exec`, `fork()`, `getpgid()`, `getpid()`, `getppid()`, `kill()`, `setpgid()`, `setsid()`35107 XBD `<sys/types.h>`, `<unistd.h>`35108 **CHANGE HISTORY**

35109 First released in Issue 1. Derived from Issue 1 of the SVID.

35110 **Issue 6**35111 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.35112 The following new requirements on POSIX implementations derive from alignment with the  
35113 Single UNIX Specification:

- 35114 • The requirement to include
- `<sys/types.h>`
- has been removed. Although
- `<sys/types.h>`
- was
- 
- 35115 required for conforming implementations of previous POSIX specifications, it was not
- 
- 35116 required for UNIX applications.

35117 **NAME**35118            **getpid** — get the process ID35119 **SYNOPSIS**

35120            #include &lt;unistd.h&gt;

35121            pid\_t getpid(void);

35122 **DESCRIPTION**35123            The *getpid()* function shall return the process ID of the calling process.35124 **RETURN VALUE**35125            The *getpid()* function shall always be successful and no return value is reserved to indicate an error.35127 **ERRORS**

35128            No errors are defined.

35129 **EXAMPLES**

35130            None.

35131 **APPLICATION USAGE**

35132            None.

35133 **RATIONALE**

35134            None.

35135 **FUTURE DIRECTIONS**

35136            None.

35137 **SEE ALSO**35138            *exec*, *fork()*, *getpgrp()*, *getppid()*, *kill()*, *mktmp()*, *setpgid()*, *setsid()*

35139            XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

35140 **CHANGE HISTORY**

35141            First released in Issue 1. Derived from Issue 1 of the SVID.

35142 **Issue 6**

35143            In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

35144            The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 35146            • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

**getpmsg()**35149 **NAME**35150 `getpmsg` — receive next message from a STREAMS file35151 **SYNOPSIS**

```
35152 OB XSI #include <stropts.h>  
35153 int getpmsg(int fildes, struct strbuf *restrict ctlptr,  
35154 struct strbuf *restrict dataptr, int *restrict bandp,  
35155 int *restrict flagsp);
```

35156 **DESCRIPTION**35157 Refer to [getmsg\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35158 **NAME**

35159           getppid — get the parent process ID

35160 **SYNOPSIS**

35161           #include &lt;unistd.h&gt;

35162           pid\_t getppid(void);

35163 **DESCRIPTION**35164           The *getppid()* function shall return the parent process ID of the calling process.35165 **RETURN VALUE**35166           The *getppid()* function shall always be successful and no return value is reserved to indicate an  
35167           error.35168 **ERRORS**

35169           No errors are defined.

35170 **EXAMPLES**

35171           None.

35172 **APPLICATION USAGE**

35173           None.

35174 **RATIONALE**

35175           None.

35176 **FUTURE DIRECTIONS**

35177           None.

35178 **SEE ALSO**35179           *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*

35180           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

35181 **CHANGE HISTORY**

35182           First released in Issue 1. Derived from Issue 1 of the SVID.

35183 **Issue 6**

35184           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

35185           The following new requirements on POSIX implementations derive from alignment with the  
35186           Single UNIX Specification:

- 35187           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
35188           required for conforming implementations of previous POSIX specifications, it was not  
35189           required for UNIX applications.

**getpriority()**35190 **NAME**35191 `getpriority, setpriority` — get and set the nice value35192 **SYNOPSIS**

```
35193 XSI      #include <sys/resource.h>
35194
35194      int getpriority(int which, id_t who);
35195      int setpriority(int which, id_t who, int value);
```

35196 **DESCRIPTION**

35197 The `getpriority()` function shall obtain the nice value of a process, process group, or user. The  
 35198 `setpriority()` function shall set the nice value of a process, process group, or user to  
 35199 `value+{NZERO}`.

35200 Target processes are specified by the values of the `which` and `who` arguments. The `which`  
 35201 argument may be one of the following values: `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`,  
 35202 indicating that the `who` argument is to be interpreted as a process ID, a process group ID, or an  
 35203 effective user ID, respectively. A 0 value for the `who` argument specifies the current process,  
 35204 process group, or user.

35205 The nice value set with `setpriority()` shall be applied to the process. If the process is multi-  
 35206 threaded, the nice value shall affect all system scope threads in the process.

35207 If more than one process is specified, `getpriority()` shall return value `{NZERO}` less than the  
 35208 lowest nice value pertaining to any of the specified processes, and `setpriority()` shall set the nice  
 35209 values of all of the specified processes to `value+{NZERO}`.

35210 The default nice value is `{NZERO}`; lower nice values shall cause more favorable scheduling.  
 35211 While the range of valid nice values is `[0, {NZERO}*2-1]`, implementations may enforce more  
 35212 restrictive limits. If `value+{NZERO}` is less than the system's lowest supported nice value,  
 35213 `setpriority()` shall set the nice value to the lowest supported value; if `value+{NZERO}` is greater  
 35214 than the system's highest supported nice value, `setpriority()` shall set the nice value to the  
 35215 highest supported value.

35216 Only a process with appropriate privileges can lower its nice value.

35217 PS|TPS Any processes or threads using `SCHED_FIFO` or `SCHED_RR` shall be unaffected by a call to  
 35218 `setpriority()`. This is not considered an error. A process which subsequently reverts to  
 35219 `SCHED_OTHER` need not have its priority affected by such a `setpriority()` call.

35220 The effect of changing the nice value may vary depending on the process-scheduling algorithm  
 35221 in effect.

35222 Since `getpriority()` can return the value `-1` upon successful completion, it is necessary to set `errno`  
 35223 to 0 prior to a call to `getpriority()`. If `getpriority()` returns the value `-1`, then `errno` can be checked  
 35224 to see if an error occurred or if the value is a legitimate nice value.

35225 **RETURN VALUE**

35226 Upon successful completion, `getpriority()` shall return an integer in the range `-{NZERO}` to  
 35227 `{NZERO}-1`. Otherwise, `-1` shall be returned and `errno` set to indicate the error.

35228 Upon successful completion, `setpriority()` shall return 0; otherwise, `-1` shall be returned and `errno`  
 35229 set to indicate the error.

35230 **ERRORS**

35231 The *getpriority()* and *setpriority()* functions shall fail if:

35232 [ESRCH] No process could be located using the *which* and *who* argument values  
35233 specified.

35234 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*  
35235 argument is not a valid process ID, process group ID, or user ID.

35236 In addition, *setpriority()* may fail if:

35237 [EPERM] A process was located, but neither the real nor effective user ID of the  
35238 executing process match the effective user ID of the process whose nice value  
35239 is being changed.

35240 [EACCES] A request was made to change the nice value to a lower numeric value and the  
35241 current process does not have appropriate privileges.

35242 **EXAMPLES**35243 **Using *getpriority()***

35244 The following example returns the current scheduling priority for the process ID returned by the  
35245 call to *getpid()*.

```
35246 #include <sys/resource.h>
35247 ...
35248 int which = PRIO_PROCESS;
35249 id_t pid;
35250 int ret;

35251 pid = getpid();
35252 ret = getpriority(which, pid);
```

35253 **Using *setpriority()***

35254 The following example sets the priority for the current process ID to -20.

```
35255 #include <sys/resource.h>
35256 ...
35257 int which = PRIO_PROCESS;
35258 id_t pid;
35259 int priority = -20;
35260 int ret;

35261 pid = getpid();
35262 ret = setpriority(which, pid, priority);
```

35263 **APPLICATION USAGE**

35264 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value  
35265  $-\{\text{NZERO}\}$ ). The nice value is in the range  $[0, 2 * \{\text{NZERO}\} - 1]$ , while the return value for  
35266 *getpriority()* and the third parameter for *setpriority()* are in the range  $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$ .

35267 **RATIONALE**

35268 None.

**getpriority()**35269 **FUTURE DIRECTIONS**

35270 None.

35271 **SEE ALSO**35272 *nice()*, *sched\_get\_priority\_max()*, *sched\_setscheduler()*35273 XBD <[sys/resource.h](#)>35274 **CHANGE HISTORY**

35275 First released in Issue 4, Version 2.

35276 **Issue 5**

35277 Moved from X/OPEN UNIX extension to BASE.

35278 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion  
35279 with functionality in the POSIX Realtime Extension.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***getprotobyname()**35280 **NAME**

35281 getprotobyname, getprotobynumber, getprotent — network protocol database functions

35282 **SYNOPSIS**

35283 #include &lt;netdb.h&gt;

35284 struct protoent \*getprotobyname(const char \*name);

35285 struct protoent \*getprotobynumber(int proto);

35286 struct protoent \*getprotoent(void);

35287 **DESCRIPTION**35288 Refer to *endprotoent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getpwent()**35289 **NAME**

35290 getpwent — get user database entry

35291 **SYNOPSIS**

```
35292 XSI #include <pwd.h>  
35293 struct passwd *getpwent(void);
```

35294 **DESCRIPTION**35295 Refer to *endpwent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35296 **NAME**35297 `getpwnam, getpwnam_r` — search user database for a name35298 **SYNOPSIS**

```
35299 #include <pwd.h>
35300 struct passwd *getpwnam(const char *name);
35301 int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
35302               size_t bufsize, struct passwd **result);
```

35303 **DESCRIPTION**35304 The `getpwnam()` function shall search the user database for an entry with a matching *name*.35305 The `getpwnam()` function need not be thread-safe.35306 Applications wishing to check for error situations should set *errno* to 0 before calling  
35307 `getpwnam()`. If `getpwnam()` returns a null pointer and *errno* is non-zero, an error occurred.

35308 The `getpwnam_r()` function shall update the **passwd** structure pointed to by *pwd* and store a  
35309 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry  
35310 from the user database with a matching *name*. Storage referenced by the structure is allocated  
35311 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to  
35312 `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either -1 without changing *errno* or an initial value  
35313 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to  
35314 by *result* on error or if the requested entry is not found.

35315 **RETURN VALUE**

35316 The `getpwnam()` function shall return a pointer to a **struct passwd** with the structure as defined  
35317 in `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested  
35318 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

35319 The return value may point to a static area which is overwritten by a subsequent call to  
35320 `getpwent()`, `getpwnam()`, or `getpwuid()`.

35321 The `getpwnam_r()` function shall return zero on success or if the requested entry was not found  
35322 and no error has occurred. If an error has occurred, an error number shall be returned to indicate  
35323 the error.

35324 **ERRORS**

35325 These functions may fail if:

35326 [EIO] An I/O error has occurred.

35327 [EINTR] A signal was caught during `getpwnam()`.

35328 [EMFILE] All file descriptors available to the process are currently open.

35329 [ENFILE] The maximum allowable number of files is currently open in the system.

35330 The `getpwnam_r()` function may fail if:

35331 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be  
35332 referenced by the resulting **passwd** structure.

**getpwnam()****EXAMPLES**

35333 Note that *sysconf*(*\_SC\_GETPW\_R\_SIZE\_MAX*) may return *-1* if there is no hard limit on the size  
 35334 of the buffer needed to store all the groups returned. This example shows how an application  
 35335 can allocate a buffer of sufficient size to work with *getpwnam\_r*(*).*  
 35336

```

35337 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
35338 size_t len;
35339 if (initlen == -1)
35340     /* Default initial length. */
35341     len = 1024;
35342 else
35343     len = (size_t) initlen;
35344 struct passwd result;
35345 struct passwd *resultp;
35346 char *buffer = malloc(len);
35347 if (buffer == NULL)
35348     ...handle error...
35349 int e;
35350 while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
35351     == ERANGE)
35352     {
35353     size_t newlen = 2 * len;
35354     if (newlen < len)
35355         ...handle error...
35356     len = newlen;
35357     char *newbuffer = realloc(buffer, len);
35358     if (newbuffer == NULL)
35359         ...handle error...
35360     buffer = newbuffer;
35361     }
35362 if (e != 0)
35363     ...handle error...
35364 free (buffer);

```

**Getting an Entry for the Login Name**

35365 The following example uses the *getlogin*(*)* function to return the name of the user who logged in;  
 35366 this information is passed to the *getpwnam*(*)* function to get the user database entry for that user.  
 35367

```

35368 #include <sys/types.h>
35369 #include <pwd.h>
35370 #include <unistd.h>
35371 #include <stdio.h>
35372 #include <stdlib.h>
35373 ...
35374 char *lgn;
35375 struct passwd *pw;
35376 ...
35377 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
35378     fprintf(stderr, "Get of user information failed.\n"); exit(1);
35379 }
35380 ...

```

35381 **APPLICATION USAGE**

35382 Three names associated with the current process can be determined: *getpwuid(getuid())* returns  
 35383 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 35384 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 35385 with the real user ID of the process.

35386 The *getpwnam\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 35387 possibly using a static data area that may be overwritten by each call.

35388 Portable applications should take into account that it is usual for an implementation to return  $-1$   
 35389 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

35390 **RATIONALE**

35391 None.

35392 **FUTURE DIRECTIONS**

35393 None.

35394 **SEE ALSO**

35395 *getpwuid()*, *sysconf()*

35396 XBD `<pwd.h>`, `<sys/types.h>`

35397 **CHANGE HISTORY**

35398 First released in Issue 1. Derived from System V Release 2.0.

35399 **Issue 5**

35400 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 35401 VALUE section.

35402 The *getpwnam\_r()* function is included for alignment with the POSIX Threads Extension.

35403 A note indicating that the *getpwnam()* function need not be reentrant is added to the  
 35404 DESCRIPTION.

35405 **Issue 6**

35406 The *getpwnam\_r()* function is marked as part of the Thread-Safe Functions option.

35407 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 35408 describing matching the *name*.

35409 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

35410 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

35411 The following new requirements on POSIX implementations derive from alignment with the  
 35412 Single UNIX Specification:

35413 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 35414 required for conforming implementations of previous POSIX specifications, it was not  
 35415 required for UNIX applications.

35416 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

35417 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

35418 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 35419 its avoidance of possibly using a static data area.

35420 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 35421 buffer from *bufsize* characters to bytes.

**getpwnam()**35422 **Issue 7**

- 35423 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 35424 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 35425 SD5-XSH-ERN-166 is applied.
- 35426 The *getpwnam\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 35427 A minor addition is made to the EXAMPLES section, reminding the application developer to
- 35428 free memory allocated as if by *malloc()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35429 **NAME**35430 `getpwuid, getpwuid_r` — search user database for a user ID35431 **SYNOPSIS**

```
35432 #include <pwd.h>
35433 struct passwd *getpwuid(uid_t uid);
35434 int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
35435 size_t bufsize, struct passwd **result);
```

35436 **DESCRIPTION**35437 The `getpwuid()` function shall search the user database for an entry with a matching *uid*.35438 The `getpwuid()` function need not be thread-safe.

35439 Applications wishing to check for error situations should set *errno* to 0 before calling `getpwuid()`.  
 35440 If `getpwuid()` returns a null pointer and *errno* is set to non-zero, an error occurred.

35441 The `getpwuid_r()` function shall update the **passwd** structure pointed to by *pwd* and store a  
 35442 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry  
 35443 from the user database with a matching *uid*. Storage referenced by the structure is allocated  
 35444 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to  
 35445 `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either `-1` without changing *errno* or an initial value  
 35446 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to  
 35447 by *result* on error or if the requested entry is not found.

35448 **RETURN VALUE**

35449 The `getpwuid()` function shall return a pointer to a **struct passwd** with the structure as defined in  
 35450 `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested entry  
 35451 is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

35452 The return value may point to a static area which is overwritten by a subsequent call to  
 35453 `getpwent()`, `getpwnam()`, or `getpwuid()`.

35454 If successful, the `getpwuid_r()` function shall return zero; otherwise, an error number shall be  
 35455 returned to indicate the error.

35456 **ERRORS**

35457 These functions may fail if:

35458 [EIO] An I/O error has occurred.

35459 [EINTR] A signal was caught during `getpwuid()`.

35460 [EMFILE] All file descriptors available to the process are currently open.

35461 [ENFILE] The maximum allowable number of files is currently open in the system.

35462 The `getpwuid_r()` function may fail if:

35463 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be  
 35464 referenced by the resulting **passwd** structure.

**getpwuid()**35465 **EXAMPLES**

35466 Note that *sysconf*(*\_SC\_GETPW\_R\_SIZE\_MAX*) may return -1 if there is no hard limit on the size  
 35467 of the buffer needed to store all the groups returned. This example shows how an application  
 35468 can allocate a buffer of sufficient size to work with *getpwuid\_r*().

```

35469     long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
35470     size_t len;
35471     if (initlen == -1)
35472         /* Default initial length. */
35473         len = 1024;
35474     else
35475         len = (size_t) initlen;
35476     struct passwd result;
35477     struct passwd *resultp;
35478     char *buffer = malloc(len);
35479     if (buffer == NULL)
35480         ...handle error...
35481     int e;
35482     while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
35483     {
35484         size_t newlen = 2 * len;
35485         if (newlen < len)
35486             ...handle error...
35487         len = newlen;
35488         char *newbuffer = realloc(buffer, len);
35489         if (newbuffer == NULL)
35490             ...handle error...
35491         buffer = newbuffer;
35492     }
35493     if (e != 0)
35494         ...handle error...
35495     free (buffer);
  
```

35496 **Getting an Entry for the Root User**

35497 The following example gets the user database entry for the user with user ID 0 (root).

```

35498     #include <sys/types.h>
35499     #include <pwd.h>
35500     ...
35501     uid_t id = 0;
35502     struct passwd *pwd;
35503     pwd = getpwuid(id);
  
```

35504 **Finding the Name for the Effective User ID**

35505 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to  
 35506 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function  
 35507 shall return the effective user ID of the calling process; this is used as the search criteria for the  
 35508 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that  
 35509 user ID value.

```
35510 #include <unistd.h>
35511 #include <sys/types.h>
35512 #include <pwd.h>
35513 ...
35514 struct passwd *pws;
35515 pws = getpwuid(geteuid());
```

35516 **Finding an Entry in the User Database**

35517 The following example uses *getpwuid()* to search the user database for a user ID that was  
 35518 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not  
 35519 found, the program prints the numeric value of the user ID for the entry.

```
35520 #include <sys/types.h>
35521 #include <pwd.h>
35522 #include <stdio.h>
35523 ...
35524 struct stat statbuf;
35525 struct passwd *pwd;
35526 ...
35527 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
35528     printf(" %-8.8s", pwd->pw_name);
35529 else
35530     printf(" %-8d", statbuf.st_uid);
```

35531 **APPLICATION USAGE**

35532 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 35533 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 35534 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 35535 with the real user ID of the process.

35536 The *getpwuid\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 35537 possibly using a static data area that may be overwritten by each call.

35538 Portable applications should take into account that it is usual for an implementation to return `-1`  
 35539 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

35540 **RATIONALE**

35541 None.

35542 **FUTURE DIRECTIONS**

35543 None.

35544 **SEE ALSO**

35545 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, *sysconf()*

35546 XBD `<pwd.h>`, `<sys/types.h>`

## getpwuid()

35547 **CHANGE HISTORY**

35548 First released in Issue 1. Derived from System V Release 2.0.

35549 **Issue 5**

35550 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
35551 VALUE section.

35552 The *getpwuid\_r()* function is included for alignment with the POSIX Threads Extension.

35553 A note indicating that the *getpwuid()* function need not be reentrant is added to the  
35554 DESCRIPTION.

35555 **Issue 6**

35556 The *getpwuid\_r()* function is marked as part of the Thread-Safe Functions option.

35557 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
35558 describing matching the *uid*.

35559 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

35560 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

35561 The following new requirements on POSIX implementations derive from alignment with the  
35562 Single UNIX Specification:

35563 

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
35564 required for conforming implementations of previous POSIX specifications, it was not  
35565 required for UNIX applications.

35566 

- In the RETURN VALUE section, the requirement to set *errno* on error is added.

35567 

- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

35568 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
35569 its avoidance of possibly using a static data area.

35570 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
35571 buffer from *bufsize* characters to bytes.

35572 **Issue 7**

35573 Austin Group Interpretation 1003.1-2001 #156 is applied.

35574 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

35575 SD5-XSH-ERN-166 is applied.

35576 The *getpwuid\_r()* function is moved from the Thread-Safe Functions option to the Base.

35577 A minor addition is made to the EXAMPLES section, reminding the application developer to  
35578 free memory allocated as if by *malloc()*.

35579 **NAME**

35580 getrlimit, setrlimit — control maximum resource consumption

35581 **SYNOPSIS**

```
35582 XSI #include <sys/resource.h>
35583 int getrlimit(int resource, struct rlimit *rlp);
35584 int setrlimit(int resource, const struct rlimit *rlp);
```

35585 **DESCRIPTION**

35586 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption  
 35587 of a variety of resources.

35588 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as  
 35589 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim\_cur*  
 35590 member specifies the current or soft limit and the *rlim\_max* member specifies the maximum or  
 35591 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the  
 35592 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or  
 35593 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both  
 35594 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints  
 35595 described above.

35596 The value RLIM\_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than  
 35597 any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the  
 35598 implementation shall not enforce limits on that resource. Specifying RLIM\_INFINITY as any  
 35599 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource  
 35600 limit.

35601 The following resources are defined:

35602	RLIMIT_CORE	This is the maximum size of a <b>core</b> file, in bytes, that may be created by a process. A limit of 0 shall prevent the creation of a <b>core</b> file. If this limit is exceeded, the writing of a <b>core</b> file shall terminate at this size.
35603		
35604		
35605	RLIMIT_CPU	This is the maximum amount of CPU time, in seconds, used by a process. If this limit is exceeded, SIGXCPU shall be generated for the process. If the process is catching or ignoring SIGXCPU, or all threads belonging to that process are blocking SIGXCPU, the behavior is unspecified.
35606		
35607		
35608		
35609	RLIMIT_DATA	This is the maximum size of a data segment of the process, in bytes. If this limit is exceeded, the <i>malloc()</i> function shall fail with <i>errno</i> set to [ENOMEM].
35610		
35611		
35612	RLIMIT_FSIZE	This is the maximum size of a file, in bytes, that may be created by a process. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ shall be generated for the thread. If the thread is blocking, or the process is catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit shall fail with <i>errno</i> set to [EFBIG].
35613		
35614		
35615		
35616		
35617		
35618	RLIMIT_NOFILE	This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate a file descriptor shall fail with <i>errno</i> set to [EMFILE]. This limit constrains the number of file descriptors that a process may allocate.
35619		
35620		
35621		
35622		

**getrlimit()**

35623	RLIMIT_STACK	This is the maximum size of the initial thread's stack, in bytes. The implementation does not automatically grow the stack beyond this limit.
35624		If this limit is exceeded, SIGSEGV shall be generated for the thread. If the
35625		thread is blocking SIGSEGV, or the process is ignoring or catching
35626		SIGSEGV and has not made arrangements to use an alternate stack, the
35627		disposition of SIGSEGV shall be set to SIG_DFL before it is generated.
35628		
35629	RLIMIT_AS	This is the maximum size of total available memory of the process, in
35630		bytes. If this limit is exceeded, the <i>malloc()</i> and <i>mmap()</i> functions shall fail
35631		with <i>errno</i> set to [ENOMEM]. In addition, the automatic stack growth
35632		fails with the effects outlined above.
35633		When using the <i>getrlimit()</i> function, if a resource limit can be represented correctly in an object
35634		of type <b>rlim_t</b> , then its representation is returned; otherwise, if the value of the resource limit is
35635		equal to that of the corresponding saved hard limit, the value returned shall be
35636		RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.
35637		When using the <i>setrlimit()</i> function, if the requested new limit is RLIM_INFINITY, the new limit
35638		shall be "no limit"; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit
35639		shall be the corresponding saved hard limit; otherwise, if the requested new limit is
35640		RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the
35641		new limit shall be the requested value. In addition, if the corresponding saved limit can be
35642		represented correctly in an object of type <b>rlim_t</b> then it shall be overwritten with the new limit.
35643		The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless
35644		a previous call to <i>getrlimit()</i> returned that value as the soft or hard limit for the corresponding
35645		resource limit.
35646		The determination of whether a limit can be correctly represented in an object of type <b>rlim_t</b> is
35647		implementation-defined. For example, some implementations permit a limit whose value is
35648		greater than RLIM_INFINITY and others do not.
35649		The <i>exec</i> family of functions shall cause resource limits to be saved.
35650	<b>RETURN VALUE</b>	
35651		Upon successful completion, <i>getrlimit()</i> and <i>setrlimit()</i> shall return 0. Otherwise, these functions
35652		shall return -1 and set <i>errno</i> to indicate the error.
35653	<b>ERRORS</b>	
35654		The <i>getrlimit()</i> and <i>setrlimit()</i> functions shall fail if:
35655	[EINVAL]	An invalid <i>resource</i> was specified; or in a <i>setrlimit()</i> call, the new <i>rlim_cur</i>
35656		exceeds the new <i>rlim_max</i> .
35657	[EPERM]	The limit specified to <i>setrlimit()</i> would have raised the maximum limit value,
35658		and the calling process does not have appropriate privileges.
35659		The <i>setrlimit()</i> function may fail if:
35660	[EINVAL]	The limit specified cannot be lowered because current usage is already higher
35661		than the limit.

35662 **EXAMPLES**

35663 None.

35664 **APPLICATION USAGE**35665 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the value  
35666 of {\_POSIX\_OPEN\_MAX} from <limits.h>, unexpected behavior may occur.35667 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the  
35668 highest currently open file descriptor +1, unexpected behavior may occur.35669 **RATIONALE**35670 It should be noted that RLIMIT\_STACK applies “at least” to the stack of the initial thread in the  
35671 process, and not to the sum of all the stacks in the process, as that would be very limiting unless  
35672 the value is so big as to provide no value at all with a single thread.35673 **FUTURE DIRECTIONS**

35674 None.

35675 **SEE ALSO**35676 *exec, fork(), malloc(), open(), sigaltstack(), sysconf(), ulimit()*

35677 XBD &lt;stropts.h&gt;, &lt;sys/resource.h&gt;

35678 **CHANGE HISTORY**

35679 First released in Issue 4, Version 2.

35680 **Issue 5**

35681 Moved from X/OPEN UNIX extension to BASE.

35682 An APPLICATION USAGE section is added.

35683 Large File Summit extensions are added.

35684 **Issue 6**35685 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for  
35686 RLIMIT\_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing  
35687 with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of  
35688 a process attempting to set the hard or soft limit for RLIMIT\_NOFILE less than the highest  
35689 currently open file descriptor +1.35690 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of  
35691 RLIMIT\_STACK in the DESCRIPTION from “the maximum size of a process stack” to “the  
35692 maximum size of the initial thread’s stack”. Text is added to the RATIONALE section.

**getrusage()**35693 **NAME**35694            **getrusage** — get information about resource utilization35695 **SYNOPSIS**

```
35696 XSI      #include <sys/resource.h>
35697         int getrusage(int who, struct rusage *r_usage);
```

35698 **DESCRIPTION**

35699        The *getrusage()* function shall provide measures of the resources used by the current process or  
35700        its terminated and waited-for child processes. If the value of the *who* argument is  
35701        RUSAGE\_SELF, information shall be returned about resources used by the current process. If the  
35702        value of the *who* argument is RUSAGE\_CHILDREN, information shall be returned about  
35703        resources used by the terminated and waited-for children of the current process. If the child is  
35704        never waited for (for example, if the parent has SA\_NOCLDWAIT set or sets SIGCHLD to  
35705        SIG\_IGN), the resource information for the child process is discarded and not included in the  
35706        resource information provided by *getrusage()*.

35707        The *r\_usage* argument is a pointer to an object of type **struct rusage** in which the returned  
35708        information is stored.

35709 **RETURN VALUE**

35710        Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*  
35711        set to indicate the error.

35712 **ERRORS**

35713        The *getrusage()* function shall fail if:

35714        [EINVAL]        The value of the *who* argument is not valid.

35715 **EXAMPLES**35716        **Using getrusage()**

35717        The following example returns information about the resources used by the current process.

```
35718     #include <sys/resource.h>
35719     ...
35720     int who = RUSAGE_SELF;
35721     struct rusage usage;
35722     int ret;
35723
35723     ret = getrusage(who, &usage);
```

35724 **APPLICATION USAGE**

35725        None.

35726 **RATIONALE**

35727        None.

35728 **FUTURE DIRECTIONS**

35729        None.

35730 **SEE ALSO**

35731        *exit()*, *sigaction()*, *time()*, *times()*, *wait()*

35732        XBD [<sys/resource.h>](#)

35733 **CHANGE HISTORY**

35734 First released in Issue 4, Version 2.

35735 **Issue 5**

35736 Moved from X/OPEN UNIX extension to BASE.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**gets()**35737 **NAME**35738 gets — get a string from a *stdin* stream35739 **SYNOPSIS**

```
35740 OB #include <stdio.h>
35741 char *gets(char *s);
```

35742 **DESCRIPTION**

35743 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 35744 conflict between the requirements described here and the ISO C standard is unintentional. This  
 35745 volume of POSIX.1-2008 defers to the ISO C standard.

35746 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed  
 35747 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall  
 35748 be discarded and a null byte shall be placed immediately after the last byte read into the array.

35749 CX The *gets()* function may mark the last data access timestamp of the file associated with *stream* for  
 35750 update. The last data access timestamp shall be marked for update by the first successful  
 35751 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or  
 35752 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

35753 **RETURN VALUE**

35754 Upon successful completion, *gets()* shall return *s*. If the end-of-file indicator for the stream is set,  
 35755 or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *gets()*  
 35756 shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set,  
 35757 CX *gets()* shall return a null pointer, and set *errno* to indicate the error.

35758 **ERRORS**35759 Refer to *fgetc()*.35760 **EXAMPLES**

35761 None.

35762 **APPLICATION USAGE**

35763 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of  
 35764 *fgets()* is recommended.

35765 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is  
 35766 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in  
 35767 such a way as to cause applications to fail, or possible system security violations.

35768 Applications should use the *fgets()* function instead of the obsolescent *gets()* function.

35769 **RATIONALE**

35770 The standard developers decided to mark the *gets()* function as obsolescent even though it is in  
 35771 the ISO C standard due to the possibility of buffer overflow.

35772 **FUTURE DIRECTIONS**35773 The *gets()* function may be removed in a future version.35774 **SEE ALSO**35775 *feof()*, *ferror()*, *fgets()*

35776 XBD &lt;stdio.h&gt;

35777 **CHANGE HISTORY**

35778 First released in Issue 1. Derived from Issue 1 of the SVID.

35779 **Issue 6**

35780 Extensions beyond the ISO C standard are marked.

35781 **Issue 7**

35782 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

35783 The *gets()* function is marked obsolescent.

35784 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getservbyname()***System Interfaces*35785 **NAME**

35786        getservbyname, getservbyport, getservent — network services database functions

35787 **SYNOPSIS**

35788        #include &lt;netdb.h&gt;

35789        struct servent \*getservbyname(const char \*name, const char \*proto);

35790        struct servent \*getservbyport(int port, const char \*proto);

35791        struct servent \*getservent(void);

35792 **DESCRIPTION**35793        Refer to *endservent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

35794 **NAME**35795        *getsid* — get the process group ID of a session leader35796 **SYNOPSIS**

35797        #include &lt;unistd.h&gt;

35798        pid\_t getsid(pid\_t *pid*);35799 **DESCRIPTION**35800        The *getsid()* function shall obtain the process group ID of the process that is the session leader of  
35801        the process specified by *pid*. If *pid* is (**pid\_t**)0, it specifies the calling process.35802 **RETURN VALUE**35803        Upon successful completion, *getsid()* shall return the process group ID of the session leader of  
35804        the specified process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.35805 **ERRORS**35806        The *getsid()* function shall fail if:35807        [EPERM]        The process specified by *pid* is not in the same session as the calling process,  
35808        and the implementation does not allow access to the process group ID of the  
35809        session leader of that process from the calling process.35810        [ESRCH]        There is no process with a process ID equal to *pid*.35811 **EXAMPLES**

35812        None.

35813 **APPLICATION USAGE**

35814        None.

35815 **RATIONALE**

35816        None.

35817 **FUTURE DIRECTIONS**

35818        None.

35819 **SEE ALSO**35820        *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*

35821        XBD &lt;unistd.h&gt;

35822 **CHANGE HISTORY**

35823        First released in Issue 4, Version 2.

35824 **Issue 5**

35825        Moved from X/OPEN UNIX extension to BASE.

35826 **Issue 7**35827        The *getsid()* function is moved from the XSI option to the Base.

**getsockname()**35828 **NAME**35829 `getsockname` — get the socket name35830 **SYNOPSIS**35831 `#include <sys/socket.h>`35832 `int getsockname(int socket, struct sockaddr *restrict address,`  
35833 `socklen_t *restrict address_len);`35834 **DESCRIPTION**35835 The `getsockname()` function shall retrieve the locally-bound name of the specified socket, store  
35836 this address in the **sockaddr** structure pointed to by the `address` argument, and store the length of  
35837 this address in the object pointed to by the `address_len` argument.35838 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
35839 the stored address shall be truncated.35840 If the socket has not been bound to a local name, the value stored in the object pointed to by  
35841 `address` is unspecified.35842 **RETURN VALUE**35843 Upon successful completion, 0 shall be returned, the `address` argument shall point to the address  
35844 of the socket, and the `address_len` argument shall point to the length of the address. Otherwise, -1  
35845 shall be returned and `errno` set to indicate the error.35846 **ERRORS**35847 The `getsockname()` function shall fail if:35848 [EBADF] The `socket` argument is not a valid file descriptor.35849 [ENOTSOCK] The `socket` argument does not refer to a socket.

35850 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

35851 The `getsockname()` function may fail if:

35852 [EINVAL] The socket has been shut down.

35853 [ENOBUFS] Insufficient resources were available in the system to complete the function.

35854 **EXAMPLES**

35855 None.

35856 **APPLICATION USAGE**

35857 None.

35858 **RATIONALE**

35859 None.

35860 **FUTURE DIRECTIONS**

35861 None.

35862 **SEE ALSO**35863 `accept()`, `bind()`, `getpeername()`, `socket()`35864 XBD `<sys/socket.h>`

**CHANGE HISTORY**

- |       |   |
|-------|---|
| 35865 | First released in Issue 6. Derived from the XNS, Issue 5.2 specification.                         |
| 35866 |   |
| 35867 | The <b>restrict</b> keyword is added to the <i>getsockname()</i> prototype for alignment with the |
| 35868 | ISO/IEC 9899:1999 standard.   |

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getsockopt()**35869 **NAME**35870 `getsockopt` — get the socket options35871 **SYNOPSIS**

```
35872 #include <sys/socket.h>
35873
35873 int getsockopt(int socket, int level, int option_name,
35874               void *restrict option_value, socklen_t *restrict option_len);
```

35875 **DESCRIPTION**35876 The `getsockopt()` function manipulates options associated with a socket.

35877 The `getsockopt()` function shall retrieve the value for the option specified by the `option_name`  
 35878 argument for the socket specified by the `socket` argument. If the size of the option value is greater  
 35879 than `option_len`, the value stored in the object pointed to by the `option_value` argument shall be  
 35880 silently truncated. Otherwise, the object pointed to by the `option_len` argument shall be modified  
 35881 to indicate the actual length of the value.

35882 The `level` argument specifies the protocol level at which the option resides. To retrieve options at  
 35883 the socket level, specify the `level` argument as `SOL_SOCKET`. To retrieve options at other levels,  
 35884 supply the appropriate level identifier for the protocol controlling the option. For example, to  
 35885 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set `level` to  
 35886 `IPPROTO_TCP` as defined in the `<netinet/in.h>` header.

35887 The socket in use may require the process to have appropriate privileges to use the `getsockopt()`  
 35888 function.

35889 The `option_name` argument specifies a single option to be retrieved. It can be one of the socket-  
 35890 level options defined in `<sys/socket.h>` and described in Section 2.10.16 (on page 522).

35891 **RETURN VALUE**

35892 Upon successful completion, `getsockopt()` shall return 0; otherwise, `-1` shall be returned and `errno`  
 35893 set to indicate the error.

35894 **ERRORS**35895 The `getsockopt()` function shall fail if:

- 35896 [EBADF] The `socket` argument is not a valid file descriptor.
- 35897 [EINVAL] The specified option is invalid at the specified socket level.
- 35898 [ENOPROTOOPT]
- 35899 The option is not supported by the protocol.
- 35900 [ENOTSOCK] The `socket` argument does not refer to a socket.

35901 The `getsockopt()` function may fail if:

- 35902 [EACCES] The calling process does not have appropriate privileges.
- 35903 [EINVAL] The socket has been shut down.
- 35904 [ENOBUFS] Insufficient resources are available in the system to complete the function.

35905 **EXAMPLES**

35906 None.

35907 **APPLICATION USAGE**

35908 None.

35909 **RATIONALE**

35910 None.

35911 **FUTURE DIRECTIONS**

35912 None.

35913 **SEE ALSO**35914 [Section 2.10.16](#) (on page 522), *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*35915 XBD [<sys/socket.h>](#), [<netinet/in.h>](#)35916 **CHANGE HISTORY**

35917 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35918 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the  
35919 ISO/IEC 9899:1999 standard.35920 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of  
35921 SO\_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.35922 **Issue 7**35923 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options  
35924 that is now in [Section 2.10.16](#) (on page 522).

**getsubopt()**35925 **NAME**35926 `getsubopt` — parse suboption arguments from a string35927 **SYNOPSIS**35928 `#include <stdlib.h>`35929 `int getsubopt(char **optionp, char * const *keylistp, char **valuep);`35930 **DESCRIPTION**35931 The `getsubopt()` function shall parse suboption arguments in a flag argument. Such options often  
35932 result from the use of `getopt()`.35933 The `getsubopt()` argument *optionp* is a pointer to a pointer to the option argument string. The  
35934 suboption arguments shall be separated by <comma> characters and each may consist of either  
35935 a single token, or a token-value pair separated by an <equals-sign>.35936 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified  
35937 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in  
35938 \**optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not  
35939 appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates  
35940 a token from its value, the application should not include an <equals-sign> in any of the strings  
35941 pointed to by *keylistp*.35942 The *valuep* argument is the address of a value string pointer.35943 If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma>  
35944 characters have been processed, if there are one or more <equals-sign> characters in a suboption  
35945 string, the first <equals-sign> in any suboption string shall be interpreted as a separator between  
35946 a token and a value. Subsequent <equals-sign> characters in a suboption string shall be  
35947 interpreted as part of the value.35948 If the string at \**optionp* contains only one suboption argument (equivalently, no <comma>  
35949 characters), `getsubopt()` shall update \**optionp* to point to the null character at the end of the  
35950 string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator  
35951 with a null character, and shall update \**optionp* to point to the start of the next suboption  
35952 argument. If the suboption argument has an associated value (equivalently, contains an <equals-  
35953 sign>), `getsubopt()` shall update \**valuep* to point to the value's first character. Otherwise, it shall  
35954 set \**valuep* to a null pointer. The calling application may use this information to determine  
35955 whether the presence or absence of a value for the suboption is an error.35956 Additionally, when `getsubopt()` fails to match the suboption argument with a token in the *keylistp*  
35957 array, the calling application should decide if this is an error, or if the unrecognized option  
35958 should be processed in another way.35959 **RETURN VALUE**35960 The `getsubopt()` function shall return the index of the matched token string, or -1 if no token  
35961 strings were matched.35962 **ERRORS**

35963 No errors are defined.

35964 **EXAMPLES**

```

35965     #include <stdio.h>
35966     #include <stdlib.h>

35967     int do_all;
35968     const char *type;
35969     int read_size;
35970     int write_size;
35971     int read_only;

35972     enum
35973     {
35974         RO_OPTION = 0,
35975         RW_OPTION,
35976         READ_SIZE_OPTION,
35977         WRITE_SIZE_OPTION
35978     };

35979     const char *mount_opts[] =
35980     {
35981         [RO_OPTION] = "ro",
35982         [RW_OPTION] = "rw",
35983         [READ_SIZE_OPTION] = "rsize",
35984         [WRITE_SIZE_OPTION] = "wsize",
35985         NULL
35986     };

35987     int
35988     main(int argc, char *argv[])
35989     {
35990         char *subopts, *value;
35991         int opt;

35992         while ((opt = getopt(argc, argv, "at:o:")) != -1)
35993             switch(opt)
35994             {
35995                 case 'a':
35996                     do_all = 1;
35997                     break;
35998                 case 't':
35999                     type = optarg;
36000                     break;
36001                 case 'o':
36002                     subopts = optarg;
36003                     while (*subopts != '\0')
36004                         switch(getsubopt(&subopts, mount_opts, &value))
36005                         {
36006                             case RO_OPTION:
36007                                 read_only = 1;
36008                                 break;
36009                             case RW_OPTION:
36010                                 read_only = 0;
36011                                 break;
36012                             case READ_SIZE_OPTION:

```

**getsubopt()**

```

36013         if (value == NULL)
36014             abort();
36015         read_size = atoi(value);
36016         break;
36017     case WRITE_SIZE_OPTION:
36018         if (value == NULL)
36019             abort();
36020         write_size = atoi(value);
36021         break;
36022     default:
36023         /* Unknown suboption. */
36024         printf("Unknown suboption '%s'\n", value);
36025         break;
36026     }
36027     break;
36028     default:
36029         abort();
36030     }
36031     /* Do the real work. */
36032     return 0;
36033 }

```

**Parsing Suboptions**

The following example uses the `getsubopt()` function to parse a *value* argument in the *optarg* external variable returned by a call to `getopt()`.

```

36037 #include <stdlib.h>
36038 ...
36039 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
36040 char *value;
36041 int opt, index;
36042 while ((opt = getopt(argc, argv, "e:")) != -1) {
36043     switch(opt) {
36044         case 'e':
36045             while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
36046                 switch(index) {
36047                     ...
36048                 }
36049                 break;
36050             }
36051     }
36052 }
36053 ...

```

**APPLICATION USAGE**

None.

36056 **RATIONALE**

36057 None.

36058 **FUTURE DIRECTIONS**

36059 None.

36060 **SEE ALSO**36061 [getopt\(\)](#)36062 XBD [<stdlib.h>](#)36063 **CHANGE HISTORY**

36064 First released in Issue 4, Version 2.

36065 **Issue 5**

36066 Moved from X/OPEN UNIX extension to BASE.

36067 **Issue 6**36068 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error  
36069 in the SYNOPSIS.36070 **Issue 7**36071 The *getsubopt()* function is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**gettimeofday()**36072 **NAME**

36073           gettimeofday — get the date and time

36074 **SYNOPSIS**

```
36075 OB XSI #include <sys/time.h>
36076 int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

36077 **DESCRIPTION**

36078       The *gettimeofday()* function shall obtain the current time, expressed as seconds and microseconds  
 36079       since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The resolution of the  
 36080       system clock is unspecified.

36081       If *tzp* is not a null pointer, the behavior is unspecified.

36082 **RETURN VALUE**

36083       The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

36084 **ERRORS**

36085       No errors are defined.

36086 **EXAMPLES**

36087       None.

36088 **APPLICATION USAGE**

36089       Applications should use the *clock\_gettime()* function instead of the obsolescent *gettimeofday()*  
 36090       function.

36091 **RATIONALE**

36092       None.

36093 **FUTURE DIRECTIONS**

36094       The *gettimeofday()* function may be removed in a future version.

36095 **SEE ALSO**

36096       *clock\_getres()*, *ctime()*

36097       XBD <sys/time.h>

36098 **CHANGE HISTORY**

36099       First released in Issue 4, Version 2.

36100 **Issue 5**

36101       Moved from X/OPEN UNIX extension to BASE.

36102 **Issue 6**

36103       The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
 36104       00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
 36105       functions.

36106       The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the  
 36107       ISO/IEC 9899:1999 standard.

36108 **Issue 7**

36109       The *gettimeofday()* function is marked obsolescent.

36110 **NAME**

36111           getuid — get a real user ID

36112 **SYNOPSIS**

36113           #include &lt;unistd.h&gt;

36114           uid\_t getuid(void);

36115 **DESCRIPTION**36116           The *getuid()* function shall return the real user ID of the calling process.36117 **RETURN VALUE**36118           The *getuid()* function shall always be successful and no return value is reserved to indicate the error.36120 **ERRORS**

36121           No errors are defined.

36122 **EXAMPLES**36123           **Setting the Effective User ID to the Real User ID**

36124           The following example sets the effective user ID and the real user ID of the current process to the real user ID of the caller.

```

36126           #include <unistd.h>
36127           #include <sys/types.h>
36128           ...
36129           setreuid(getuid(), getuid());
36130           ...

```

36131 **APPLICATION USAGE**

36132           None.

36133 **RATIONALE**

36134           None.

36135 **FUTURE DIRECTIONS**

36136           None.

36137 **SEE ALSO**36138           *getegid(), geteuid(), getgid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

36139           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

36140 **CHANGE HISTORY**

36141           First released in Issue 1. Derived from Issue 1 of the SVID.

36142 **Issue 6**

36143           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

36144           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 36146           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

**getutxent()**36149 **NAME**

36150 getutxent, getutxid, getutxline — get user accounting database entries

36151 **SYNOPSIS**

```
36152 XSI #include <utmpx.h>
36153 struct utmpx *getutxent(void);
36154 struct utmpx *getutxid(const struct utmpx *id);
36155 struct utmpx *getutxline(const struct utmpx *line);
```

36156 **DESCRIPTION**36157 Refer to *endutxent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

36158 **NAME**36159 `getwc` — get a wide character from a stream36160 **SYNOPSIS**36161 `#include <stdio.h>`36162 `#include <wchar.h>`36163 `wint_t getwc(FILE *stream);`36164 **DESCRIPTION**

36165 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 36166 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36167 volume of POSIX.1-2008 defers to the ISO C standard.

36168 The `getwc()` function shall be equivalent to `fgetwc()`, except that if it is implemented as a macro it  
 36169 may evaluate `stream` more than once, so the argument should never be an expression with side-  
 36170 effects.

36171 **RETURN VALUE**36172 Refer to `fgetwc()`.36173 **ERRORS**36174 Refer to `fgetwc()`.36175 **EXAMPLES**

36176 None.

36177 **APPLICATION USAGE**

36178 Since it may be implemented as a macro, `getwc()` may treat incorrectly a `stream` argument with  
 36179 side-effects. In particular, `getwc(*f++)` does not necessarily work as expected. Therefore, use of  
 36180 this function is not recommended; `fgetwc()` should be used instead.

36181 **RATIONALE**

36182 None.

36183 **FUTURE DIRECTIONS**

36184 None.

36185 **SEE ALSO**36186 `fgetwc()`36187 XBD `<stdio.h>`, `<wchar.h>`36188 **CHANGE HISTORY**

36189 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
 36190 draft.

36191 **Issue 5**36192 The Optional Header (OH) marking is removed from `<stdio.h>`.

## getwchar()

### 36193 NAME

36194 `getwchar` — get a wide character from a *stdin* stream

### 36195 SYNOPSIS

36196 `#include <wchar.h>`

36197 `wint_t getwchar(void);`

### 36198 DESCRIPTION

36199 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
36200 conflict between the requirements described here and the ISO C standard is unintentional. This  
36201 volume of POSIX.1-2008 defers to the ISO C standard.

36202 The `getwchar()` function shall be equivalent to `getwc(stdin)`.

### 36203 RETURN VALUE

36204 Refer to `fgetwc()`.

### 36205 ERRORS

36206 Refer to `fgetwc()`.

### 36207 EXAMPLES

36208 None.

### 36209 APPLICATION USAGE

36210 If the `wint_t` value returned by `getwchar()` is stored into a variable of type `wchar_t` and then  
36211 compared against the `wint_t` macro `WEOF`, the result may be incorrect. Only the `wint_t` type is  
36212 guaranteed to be able to represent any wide character and `WEOF`.

### 36213 RATIONALE

36214 None.

### 36215 FUTURE DIRECTIONS

36216 None.

### 36217 SEE ALSO

36218 `fgetwc()`, `getwc()`

36219 XBD `<wchar.h>`

### 36220 CHANGE HISTORY

36221 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
36222 draft.

36223 **NAME**

36224 glob, globfree — generate pathnames matching a pattern

36225 **SYNOPSIS**

```
36226 #include <glob.h>
36227 int glob(const char *restrict pattern, int flags,
36228         int (*errfunc)(const char *epath, int eerrno),
36229         glob_t *restrict pglob);
36230 void globfree(glob_t *pglob);
```

36231 **DESCRIPTION**

36232 The *glob()* function is a pathname generator that shall implement the rules defined in XCU  
 36233 Section 2.13 (on page 2332), with optional support for rule 3 in XCU Section 2.13.3 (on page  
 36234 2333).

36235 The structure type **glob\_t** is defined in **<glob.h>** and includes at least the following members:

Member Type	Member Name	Description
size_t	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .
char **	<i>gl_pathv</i>	Pointer to a list of matched pathnames.
size_t	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .

36241 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function  
 36242 shall match all accessible pathnames against this pattern and develop a list of all pathnames that  
 36243 match. In order to have access to a pathname, *glob()* requires search permission on every  
 36244 component of a path except the last, and read permission on each directory of any filename  
 36245 component of *pattern* that contains any of the following special characters: '\*', '?', and '['.

36246 The *glob()* function shall store the number of matched pathnames into *pglob->gl\_pathc* and a  
 36247 pointer to a list of pointers to pathnames into *pglob->gl\_pathv*. The pathnames shall be in sort  
 36248 order as defined by the current setting of the *LC\_COLLATE* category; see XBD Section 7.3.2 (on  
 36249 page 146). The first pointer after the last pathname shall be a null pointer. If the pattern does not  
 36250 match any pathnames, the returned number of matched paths is set to 0, and the contents of  
 36251 *pglob->gl\_pathv* are implementation-defined.

36252 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function  
 36253 shall allocate other space as needed, including the memory pointed to by *gl\_pathv*. The *globfree()*  
 36254 function shall free any space associated with *pglob* from a previous call to *glob()*.

36255 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-  
 36256 inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

36257	<b>GLOB_APPEND</b>	Append pathnames generated to the ones from a previous call to <i>glob()</i> .
36258	<b>GLOB_DOOFFS</b>	Make use of <i>pglob-&gt;gl_offs</i> . If this flag is set, <i>pglob-&gt;gl_offs</i> is used to 36259 specify how many null pointers to add to the beginning of 36260 <i>pglob-&gt;gl_pathv</i> . In other words, <i>pglob-&gt;gl_pathv</i> shall point to 36261 <i>pglob-&gt;gl_offs</i> null pointers, followed by <i>pglob-&gt;gl_pathc</i> pathname 36262 pointers, followed by a null pointer.
36263	<b>GLOB_ERR</b>	Cause <i>glob()</i> to return when it encounters a directory that it cannot open 36264 or read. Ordinarily, <i>glob()</i> continues to find matches.
36265	<b>GLOB_MARK</b>	Each pathname that is a directory that matches <i>pattern</i> shall have a 36266 <b>&lt;slash&gt;</b> appended.

**glob()**

- 36267 GLOB\_NOCHECK Supports rule 3 in XCU Section 2.13.3 (on page 2333). If *pattern* does not  
 36268 match any pathname, then *glob()* shall return a list consisting of only  
 36269 *pattern*, and the number of matched pathnames is 1.
- 36270 GLOB\_NOESCAPE Disable backslash escaping.
- 36271 GLOB\_NOSORT Ordinarily, *glob()* sorts the matching pathnames according to the current  
 36272 setting of the *LC\_COLLATE* category; see XBD Section 7.3.2 (on page 146).  
 36273 When this flag is used, the order of pathnames returned is unspecified.

36274 The GLOB\_APPEND flag can be used to append a new set of pathnames to those found in a  
 36275 previous call to *glob()*. The following rules apply to applications when two or more calls to  
 36276 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 36277 1. The first such call shall not set GLOB\_APPEND. All subsequent calls shall set it.
- 36278 2. All the calls shall set GLOB\_DOOFFS, or all shall not set it.
- 36279 3. After the second call, *pglob->gl\_pathv* points to a list containing the following:
  - 36280 a. Zero or more null pointers, as specified by GLOB\_DOOFFS and *pglob->gl\_offs*.
  - 36281 b. Pointers to the pathnames that were in the *pglob->gl\_pathv* list before the call, in  
 36282 the same order as before.
  - 36283 c. Pointers to the new pathnames generated by the second call, in the specified order.
- 36284 4. The count returned in *pglob->gl\_pathc* shall be the total number of pathnames from the  
 36285 two calls.
- 36286 5. The application can change any of the fields after a call to *glob()*. If it does, the  
 36287 application shall reset them to the original value before a subsequent call, using the same  
 36288 *pglob* value, to *globfree()* or *glob()* with the GLOB\_APPEND flag.

36289 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not  
 36290 a null pointer, *glob()* calls (*\*errfunc()*) with two arguments:

- 36291 1. The *epath* argument is a pointer to the path that failed.
- 36292 2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or  
 36293 *stat()*. (Other values may be used to report other errors not explicitly documented for  
 36294 those functions.)

36295 If (*\*errfunc()*) is called and returns non-zero, or if the GLOB\_ERR flag is set in *flags*, *glob()* shall  
 36296 stop the scan and return GLOB\_ABORTED after setting *gl\_pathc* and *gl\_pathv* in *pglob* to reflect  
 36297 the paths already scanned. If GLOB\_ERR is not set and either *errfunc* is a null pointer or  
 36298 (*\*errfunc()*) returns 0, the error shall be ignored.

36299 The *glob()* function shall not fail because of large files.

**RETURN VALUE**

36300 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl\_pathc* shall return the  
 36301 number of matched pathnames and the argument *pglob->gl\_pathv* shall contain a pointer to a  
 36302 null-terminated list of matched and sorted pathnames. However, if *pglob->gl\_pathc* is 0, the  
 36303 content of *pglob->gl\_pathv* is undefined.

36304 The *globfree()* function shall not return a value.

36305 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in  
 36306 **<glob.h>**. The arguments *pglob->gl\_pathc* and *pglob->gl\_pathv* are still set as defined above.

36308 **ERRORS**

36309 The *glob()* function shall fail and return the corresponding value if:

- 36310 **GLOB\_ABORTED** The scan was stopped because **GLOB\_ERR** was set or (*\*errfunc()*)  
36311 returned non-zero.
- 36312 **GLOB\_NOMATCH** The pattern does not match any existing pathname, and  
36313 **GLOB\_NOCHECK** was not set in flags.
- 36314 **GLOB\_NOSPACE** An attempt to allocate memory failed.

36315 **EXAMPLES**

36316 One use of the **GLOB\_DOOFFS** flag is by applications that build an argument list for use with  
36317 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the  
36318 equivalent of:

```
36319 ls -l *.c
```

36320 but for some reason:

```
36321 system("ls -l *.c")
```

36322 is not acceptable. The application could obtain approximately the same result using the  
36323 sequence:

```
36324 globbuf.gl_offs = 2;
36325 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
36326 globbuf.gl_pathv[0] = "ls";
36327 globbuf.gl_pathv[1] = "-l";
36328 execvp("ls", &globbuf.gl_pathv[0]);
```

36329 Using the same example:

```
36330 ls -l *.c *.h
```

36331 could be approximately simulated using **GLOB\_APPEND** as follows:

```
36332 globbuf.gl_offs = 2;
36333 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
36334 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
36335 ...
```

36336 **APPLICATION USAGE**

36337 This function is not provided for the purpose of enabling utilities to perform pathname  
36338 expansion on their arguments, as this operation is performed by the shell, and utilities are  
36339 explicitly not expected to redo this. Instead, it is provided for applications that need to do  
36340 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or  
36341 read from a file.

36342 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

36343 Note that *gl\_pathc* and *gl\_pathv* have meaning even if *glob()* fails. This allows *glob()* to report  
36344 partial results in the event of an error. However, if *gl\_pathc* is 0, *gl\_pathv* is unspecified even if  
36345 *glob()* did not return an error.

36346 The **GLOB\_NOCHECK** option could be used when an application wants to expand a pathname  
36347 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility  
36348 might use this for option-arguments, for example.

36349 The new pathnames generated by a subsequent call with **GLOB\_APPEND** are not sorted  
36350 together with the previous pathnames. This mirrors the way that the shell handles pathname

**glob()**

36351 expansion when multiple expansions are done on a command line.

36352 Applications that need tilde and parameter expansion should use *wordexp()*.

**RATIONALE**

36354 It was claimed that the GLOB\_DOOFFS flag is unnecessary because it could be simulated using:

```
36355 new = (char **)malloc((n + pglob->gl_pathc + 1)
36356     * sizeof(char *));
36357 (void) memcpy(new+n, pglob->gl_pathv,
36358     pglob->gl_pathc * sizeof(char *));
36359 (void) memset(new, 0, n * sizeof(char *));
36360 free (pglob->gl_pathv);
36361 pglob->gl_pathv = new;
```

36362 However, this assumes that the memory pointed to by *gl\_pathv* is a block that was separately  
 36363 created using *malloc()*. This is not necessarily the case. An application should make no  
 36364 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have  
 36365 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have  
 36366 been created using a different memory allocator. It is not the intent of the standard developers to  
 36367 specify or imply how the memory used by *glob()* is managed.

36368 The GLOB\_APPEND flag would be used when an application wants to expand several different  
 36369 patterns into a single list.

**FUTURE DIRECTIONS**

36370 None.

**SEE ALSO**

36373 *exec*, *fdopendir()*, *fnmatch()*, *fstatat()*, *readdir()*, Section 2.6

36374 XBD Section 7.3.2 (on page 146), [<glob.h>](#)

**CHANGE HISTORY**

36376 First released in Issue 4. Derived from the ISO POSIX-2 standard.

**Issue 5**

36378 Moved from POSIX2 C-language Binding to BASE.

**Issue 6**

36380 The normative text is updated to avoid use of the term “must” for application requirements.

36381 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999  
 36382 standard.

36383 **NAME**

36384 gmtime, gmtime\_r — convert a time value to a broken-down UTC time

36385 **SYNOPSIS**

```
36386 #include <time.h>
36387 struct tm *gmtime(const time_t *timer);
36388 CX struct tm *gmtime_r(const time_t *restrict timer,
36389 struct tm *restrict result);
```

36390 **DESCRIPTION**

36391 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C  
 36392 standard. Any conflict between the requirements described here and the ISO C standard is  
 36393 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

36394 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into  
 36395 a broken-down time, expressed as Coordinated Universal Time (UTC).

36396 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime()*  
 36397 and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in  
 36398 the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page  
 36399 113), where the names in the structure and in the expression correspond.

36400 The same relationship shall apply for *gmtime\_r()*.

36401 The *gmtime()* function need not be thread-safe.

36402 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 36403 objects: a broken-down time structure and an array of type **char**. Execution of any of the  
 36404 functions may overwrite the information returned in either of these objects by any of the other  
 36405 functions.

36406 The *gmtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
 36407 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down  
 36408 time is stored in the structure referred to by *result*. The *gmtime\_r()* function shall also return the  
 36409 address of the same structure.

36410 **RETURN VALUE**

36411 Upon successful completion, the *gmtime()* function shall return a pointer to a **struct tm**. If an  
 36412 CX error is detected, *gmtime()* shall return a null pointer and set *errno* to indicate the error.

36413 Upon successful completion, *gmtime\_r()* shall return the address of the structure pointed to by  
 36414 the argument *result*. If an error is detected, *gmtime\_r()* shall return a null pointer and set *errno* to  
 36415 indicate the error.

36416 **ERRORS**

36417 CX The *gmtime()* and *gmtime\_r()* functions shall fail if:

36418 CX [EOVERFLOW] The result cannot be represented.

**gmtime()**36419 **EXAMPLES**

36420 None.

36421 **APPLICATION USAGE**36422 The *gmtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
36423 possibly using a static data area that may be overwritten by each call.36424 **RATIONALE**

36425 None.

36426 **FUTURE DIRECTIONS**

36427 None.

36428 **SEE ALSO**36429 *asctime(), clock(), ctime(), difftime(), localtime(), mktime(), strftime(), strptime(), time(), utime()*36430 XBD Section 4.15 (on page 113), <[time.h](#)>36431 **CHANGE HISTORY**

36432 First released in Issue 1. Derived from Issue 1 of the SVID.

36433 **Issue 5**36434 A note indicating that the *gmtime()* function need not be reentrant is added to the  
36435 DESCRIPTION.36436 The *gmtime\_r()* function is included for alignment with the POSIX Threads Extension.36437 **Issue 6**36438 The *gmtime\_r()* function is marked as part of the Thread-Safe Functions option.

36439 Extensions beyond the ISO C standard are marked.

36440 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
36441 its avoidance of possibly using a static data area.36442 The **restrict** keyword is added to the *gmtime\_r()* prototype for alignment with the  
36443 ISO/IEC 9899: 1999 standard.36444 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [Eoverflow]  
36445 error.36446 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling  
36447 for *gmtime\_r()*.36448 **Issue 7**

36449 Austin Group Interpretation 1003.1-2001 #156 is applied.

36450 The *gmtime\_r()* function is moved from the Thread-Safe Functions option to the Base.

36451 **NAME**36452 `grantpt` — grant access to the slave pseudo-terminal device36453 **SYNOPSIS**

```
36454 XSI #include <stdlib.h>
36455 int grantpt(int fildes);
```

36456 **DESCRIPTION**

36457 The `grantpt()` function shall change the mode and ownership of the slave pseudo-terminal  
 36458 device associated with its master pseudo-terminal counterpart. The `fildes` argument is a file  
 36459 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to  
 36460 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The  
 36461 permission mode of the slave pseudo-terminal shall be set to readable and writable by the  
 36462 owner, and writable by the group.

36463 The behavior of the `grantpt()` function is unspecified if the application has installed a signal  
 36464 handler to catch SIGCHLD signals.

36465 **RETURN VALUE**

36466 Upon successful completion, `grantpt()` shall return 0; otherwise, it shall return -1 and set `errno` to  
 36467 indicate the error.

36468 **ERRORS**

36469 The `grantpt()` function may fail if:

- |       |          |  |
|-------|----------|--|
| 36470 | [EBADF]  | The <code>fildes</code> argument is not a valid open file descriptor.                    |
| 36471 | [EINVAL] | The <code>fildes</code> argument is not associated with a master pseudo-terminal device. |
| 36472 | [EACCES] | The corresponding slave pseudo-terminal device could not be accessed.                    |

36473 **EXAMPLES**

36474 None.

36475 **APPLICATION USAGE**

36476 None.

36477 **RATIONALE**

36478 None.

36479 **FUTURE DIRECTIONS**

36480 None.

36481 **SEE ALSO**

36482 `open()`, `ptsname()`, `unlockpt()`

36483 XBD <stdlib.h>

36484 **CHANGE HISTORY**

36485 First released in Issue 4, Version 2.

36486 **Issue 5**

36487 Moved from X/OPEN UNIX extension to BASE.

36488 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

**hcreate()**36489 **NAME**36490 `hcreate`, `hdestroy`, `hsearch` — manage hash search table36491 **SYNOPSIS**

```

36492 XSI      #include <search.h>
36493
36493      int hcreate(size_t nel);
36494      void hdestroy(void);
36495      ENTRY *hsearch(ENTRY item, ACTION action);

```

36496 **DESCRIPTION**36497 The `hcreate()`, `hdestroy()`, and `hsearch()` functions shall manage hash search tables.

36498 The `hcreate()` function shall allocate sufficient space for the table, and the application shall ensure it is called before `hsearch()` is used. The `nel` argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

36502 The `hdestroy()` function shall dispose of the search table, and may be followed by another call to `hcreate()`. After the call to `hdestroy()`, the data can no longer be considered accessible.

36504 The `hsearch()` function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The `item` argument is a structure of type **ENTRY** (defined in the `<search.h>` header) containing two pointers: `item.key` points to the comparison key (a `char *`), and `item.data` (a `void *`) points to any other data to be associated with that key. The comparison function used by `hsearch()` is `strcmp()`. The `action` argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

36513 These functions need not be thread-safe.

36514 **RETURN VALUE**36515 The `hcreate()` function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.36517 The `hdestroy()` function shall not return a value.36518 The `hsearch()` function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.36520 **ERRORS**36521 The `hcreate()` and `hsearch()` functions may fail if:

36522 [ENOMEM] Insufficient storage space is available.

36523 **EXAMPLES**

36524 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```

36527      #include <stdio.h>
36528      #include <search.h>
36529      #include <string.h>
36530
36530      struct info {          /* This is the info stored in the table */
36531          int age, room;    /* other than the key. */
36532      };

```

```

36533     #define NUM_EMPL    5000    /* # of elements in search table. */
36534     int main(void)
36535     {
36536         char string_space[NUM_EMPL*20]; /* Space to store strings. */
36537         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
36538         char *str_ptr = string_space; /* Next space in string_space. */
36539         struct info *info_ptr = info_space;
36540                                     /* Next space in info_space. */
36541         ENTRY item;
36542         ENTRY *found_item; /* Name to look for in table. */
36543         char name_to_find[30];
36544
36545         int i = 0;
36546
36547         /* Create table; no error checking is performed. */
36548         (void) hcreate(NUM_EMPL);
36549         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
36550                    &info_ptr->room) != EOF && i++ < NUM_EMPL) {
36551
36552             /* Put information in structure, and structure in item. */
36553             item.key = str_ptr;
36554             item.data = info_ptr;
36555             str_ptr += strlen(str_ptr) + 1;
36556             info_ptr++;
36557
36558             /* Put item into table. */
36559             (void) hsearch(item, ENTER);
36560         }
36561
36562         /* Access table. */
36563         item.key = name_to_find;
36564         while (scanf("%s", item.key) != EOF) {
36565             if ((found_item = hsearch(item, FIND)) != NULL) {
36566                 /* If item is in the table. */
36567                 (void) printf("found %s, age = %d, room = %d\n",
36568                               found_item->key,
36569                               ((struct info *)found_item->data)->age,
36570                               ((struct info *)found_item->data)->room);
36571             }
36572             else
36573                 (void) printf("no such employee %s\n", name_to_find);
36574         }
36575         return 0;
36576     }

```

**APPLICATION USAGE**

The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**hcreate()**36577 **SEE ALSO**36578 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tdelete()*36579 XBD <[search.h](#)>36580 **CHANGE HISTORY**

36581 First released in Issue 1. Derived from Issue 1 of the SVID.

36582 **Issue 6**

36583 The normative text is updated to avoid use of the term “must” for application requirements.

36584 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

36585 **Issue 7**

36586 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

36587 **NAME**

36588        htonl, htons, ntohl, ntohs — convert values between host and network byte order

36589 **SYNOPSIS**

```
36590        #include <arpa/inet.h>
36591        uint32_t htonl(uint32_t hostlong);
36592        uint16_t htons(uint16_t hostshort);
36593        uint32_t ntohl(uint32_t netlong);
36594        uint16_t ntohs(uint16_t netshort);
```

36595 **DESCRIPTION**36596        These functions shall convert 16-bit and 32-bit quantities between network byte order and host  
36597        byte order.

36598        On some implementations, these functions are defined as macros.

36599        The `uint32_t` and `uint16_t` types are defined in `<inttypes.h>`.36600 **RETURN VALUE**36601        The `htonl()` and `htons()` functions shall return the argument value converted from host to  
36602        network byte order.36603        The `ntohl()` and `ntohs()` functions shall return the argument value converted from network to  
36604        host byte order.36605 **ERRORS**

36606        No errors are defined.

36607 **EXAMPLES**

36608        None.

36609 **APPLICATION USAGE**36610        These functions are most often used in conjunction with IPv4 addresses and ports as returned by  
36611        `gethostent()` and `getservent()`.36612 **RATIONALE**

36613        None.

36614 **FUTURE DIRECTIONS**

36615        None.

36616 **SEE ALSO**36617        `endhostent()`, `endservent()`36618        XBD `<arpa/inet.h>`, `<inttypes.h>`36619 **CHANGE HISTORY**

36620        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**hypot()**36621 **NAME**36622 `hypot, hypotf, hypotl` — Euclidean distance function36623 **SYNOPSIS**36624 `#include <math.h>`36625 `double hypot(double x, double y);`36626 `float hypotf(float x, float y);`36627 `long double hypotl(long double x, long double y);`36628 **DESCRIPTION**36629 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
36630 conflict between the requirements described here and the ISO C standard is unintentional. This  
36631 volume of POSIX.1-2008 defers to the ISO C standard.36632 These functions shall compute the value of the square root of  $x^2+y^2$  without undue overflow or  
36633 underflow.36634 An application wishing to check for error situations should set *errno* to zero and call  
36635 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
36636 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
36637 zero, an error has occurred.36638 **RETURN VALUE**36639 Upon successful completion, these functions shall return the length of the hypotenuse of a right-  
36640 angled triangle with sides of length *x* and *y*.36641 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and  
36642 *hypotl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
36643 respectively.36644 **MX** If *x* or *y* is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned (even if one of *x* or *y* is NaN).36645 If *x* or *y* is NaN, and the other is not  $\pm\text{Inf}$ , a NaN shall be returned.36646 If both arguments are subnormal and the correct result is subnormal, a range error may occur  
36647 and the correct result is returned.36648 **ERRORS**

36649 These functions shall fail if:

36650 **Range Error** The result overflows.36651 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
36652 then *errno* shall be set to [ERANGE]. If the integer expression  
36653 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
36654 floating-point exception shall be raised.

36655 These functions may fail if:

36656 **MX** **Range Error** The result underflows.36657 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
36658 then *errno* shall be set to [ERANGE]. If the integer expression  
36659 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
36660 floating-point exception shall be raised.

36661 **EXAMPLES**

36662 See the EXAMPLES section in *atan2()*.

36663 **APPLICATION USAGE**

36664 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x, -y)* are equivalent.

36665 *hypot(x, ±0)* is equivalent to *fabs(x)*.

36666 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

36668 These functions take precautions against overflow during intermediate steps of the computation.

36669 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* & MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

36671 **RATIONALE**

36672 None.

36673 **FUTURE DIRECTIONS**

36674 None.

36675 **SEE ALSO**

36676 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*

36677 XBD Section 4.19 (on page 116), <math.h>

36678 **CHANGE HISTORY**

36679 First released in Issue 1. Derived from Issue 1 of the SVID.

36680 **Issue 5**

36681 The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

36683 **Issue 6**

36684 The *hypot()* function is no longer marked as an extension.

36685 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

36687 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

36689 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

36691 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES section.

**iconv()**36693 **NAME**36694 `iconv` — codeset conversion function36695 **SYNOPSIS**

```
36696 #include <iconv.h>
36697
36697 size_t iconv(iconv_t cd, char **restrict inbuf,
36698             size_t *restrict inbytesleft, char **restrict outbuf,
36699             size_t *restrict outbytesleft);
```

36700 **DESCRIPTION**

36701 The `iconv()` function shall convert the sequence of characters from one codeset, in the array  
 36702 specified by `inbuf`, into a sequence of corresponding characters in another codeset, in the array  
 36703 specified by `outbuf`. The codesets are those specified in the `iconv_open()` call that returned the  
 36704 conversion descriptor, `cd`. The `inbuf` argument points to a variable that points to the first  
 36705 character in the input buffer and `inbytesleft` indicates the number of bytes to the end of the buffer  
 36706 to be converted. The `outbuf` argument points to a variable that points to the first available byte in  
 36707 the output buffer and `outbytesleft` indicates the number of the available bytes to the end of the  
 36708 buffer.

36709 For state-dependent encodings, the conversion descriptor `cd` is placed into its initial shift state by  
 36710 a call for which `inbuf` is a null pointer, or for which `inbuf` points to a null pointer. When `iconv()`  
 36711 is called in this way, and if `outbuf` is not a null pointer or a pointer to a null pointer, and `outbytesleft`  
 36712 points to a positive value, `iconv()` shall place, into the output buffer, the byte sequence to change  
 36713 the output buffer to its initial shift state. If the output buffer is not large enough to hold the  
 36714 entire reset sequence, `iconv()` shall fail and set `errno` to [E2BIG]. Subsequent calls with `inbuf` as  
 36715 other than a null pointer or a pointer to a null pointer cause the conversion to take place from  
 36716 the current state of the conversion descriptor.

36717 If a sequence of input bytes does not form a valid character in the specified codeset, conversion  
 36718 shall stop after the previous successfully converted character. If the input buffer ends with an  
 36719 incomplete character or shift sequence, conversion shall stop after the previous successfully  
 36720 converted bytes. If the output buffer is not large enough to hold the entire converted input,  
 36721 conversion shall stop just prior to the input bytes that would cause the output buffer to  
 36722 overflow. The variable pointed to by `inbuf` shall be updated to point to the byte following the last  
 36723 byte successfully used in the conversion. The value pointed to by `inbytesleft` shall be  
 36724 decremented to reflect the number of bytes still not converted in the input buffer. The variable  
 36725 pointed to by `outbuf` shall be updated to point to the byte following the last byte of converted  
 36726 output data. The value pointed to by `outbytesleft` shall be decremented to reflect the number of  
 36727 bytes still available in the output buffer. For state-dependent encodings, the conversion  
 36728 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully  
 36729 converted byte sequence.

36730 If `iconv()` encounters a character in the input buffer that is valid, but for which an identical  
 36731 character does not exist in the target codeset, `iconv()` shall perform an implementation-defined  
 36732 conversion on this character.

36733 **RETURN VALUE**

36734 The `iconv()` function shall update the variables pointed to by the arguments to reflect the extent  
 36735 of the conversion and return the number of non-identical conversions performed. If the entire  
 36736 string in the input buffer is converted, the value pointed to by `inbytesleft` shall be 0. If the input  
 36737 conversion is stopped due to any conditions mentioned above, the value pointed to by `inbytesleft`  
 36738 shall be non-zero and `errno` shall be set to indicate the condition. If an error occurs, `iconv()` shall  
 36739 return `(size_t)-1` and set `errno` to indicate the error.

36740 **ERRORS**36741 The *iconv()* function shall fail if:36742 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the  
36743 input codeset.

36744 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

36745 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at  
36746 the end of the input buffer.36747 The *iconv()* function may fail if:36748 [EBADF] The *cd* argument is not a valid open conversion descriptor.36749 **EXAMPLES**

36750 None.

36751 **APPLICATION USAGE**36752 The *inbuf* argument indirectly points to the memory area which contains the conversion input  
36753 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of  
36754 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to  
36755 containing data that is directly representable in the ISO C standard language **char** data type. The  
36756 type of *inbuf* and *outbuf*, **char \*\***, does not imply that the objects pointed to are interpreted as  
36757 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that  
36758 represents a character in a given character set encoding scheme is done internally within the  
36759 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can  
36760 contain all zero octets that are not interpreted as string terminators but as coded character data  
36761 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**,  
36762 and so on) read or stored in the objects is not specified, but may be inferred for both the input and  
36763 output data by the converters determined by the *fromcode* and *toencode* arguments of *iconv\_open()*.36764 Regardless of the data type inferred by the converter, the size of the remaining space in both  
36765 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.36766 For implementations that support the conversion of state-dependent encodings, the conversion  
36767 descriptor must be able to accurately reflect the shift-state in effect at the end of the last  
36768 successful conversion. It is not required that the conversion descriptor itself be updated, which  
36769 would require it to be a pointer type. Thus, implementations are free to implement the  
36770 descriptor as a handle (other than a pointer type) by which the conversion information can be  
36771 accessed and updated.36772 **RATIONALE**

36773 None.

36774 **FUTURE DIRECTIONS**

36775 None.

36776 **SEE ALSO**36777 *iconv\_open()*, *iconv\_close()*, *mbsrtowcs()*36778 XBD <**iconv.h**>36779 **CHANGE HISTORY**

36780 First released in Issue 4. Derived from the HP-UX Manual.

**iconv()**36781 **Issue 6**

36782 The SYNOPSIS has been corrected to align with the `<iconv.h>` reference page.

36783 The **restrict** keyword is added to the `iconv()` prototype for alignment with the  
36784 ISO/IEC 9899:1999 standard.

36785 **Issue 7**

36786 The `iconv()` function is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

36787 **NAME**36788 `iconv_close` — codeset conversion deallocation function36789 **SYNOPSIS**36790 `#include <iconv.h>`36791 `int iconv_close(iconv_t cd);`36792 **DESCRIPTION**36793 The `iconv_close()` function shall deallocate the conversion descriptor `cd` and all other associated  
36794 resources allocated by `iconv_open()`.36795 If a file descriptor is used to implement the type **iconv\_t**, that file descriptor shall be closed.36796 **RETURN VALUE**36797 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and `errno` set to  
36798 indicate the error.36799 **ERRORS**36800 The `iconv_close()` function may fail if:

36801 [EBADF] The conversion descriptor is invalid.

36802 **EXAMPLES**

36803 None.

36804 **APPLICATION USAGE**

36805 None.

36806 **RATIONALE**

36807 None.

36808 **FUTURE DIRECTIONS**

36809 None.

36810 **SEE ALSO**36811 `iconv()`, `iconv_open()`36812 XBD `<iconv.h>`36813 **CHANGE HISTORY**

36814 First released in Issue 4. Derived from the HP-UX Manual.

36815 **Issue 7**36816 The `iconv_close()` function is moved from the XSI option to the Base.

**iconv\_open()**36817 **NAME**

36818 iconv\_open — codeset conversion allocation function

36819 **SYNOPSIS**

36820 #include &lt;iconv.h&gt;

36821 iconv\_t iconv\_open(const char \*tocode, const char \*fromcode);

36822 **DESCRIPTION**

36823 The *iconv\_open()* function shall return a conversion descriptor that describes a conversion from  
 36824 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified  
 36825 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion  
 36826 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with  
 36827 *iconv()*.

36828 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.36829 A conversion descriptor shall remain valid until it is closed by *iconv\_close()* or an implicit close.36830 If a file descriptor is used to implement conversion descriptors, the FD\_CLOEXEC flag shall be  
 36831 set; see <fcntl.h>.36832 **RETURN VALUE**

36833 Upon successful completion, *iconv\_open()* shall return a conversion descriptor for use on  
 36834 subsequent calls to *iconv()*. Otherwise, *iconv\_open()* shall return (**iconv\_t**)-1 and set *errno* to  
 36835 indicate the error.

36836 **ERRORS**36837 The *iconv\_open()* function may fail if:

36838 [EMFILE] All file descriptors available to the process are currently open.

36839 [ENFILE] Too many files are currently open in the system.

36840 [ENOMEM] Insufficient storage space is available.

36841 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the  
 36842 implementation.36843 **EXAMPLES**

36844 None.

36845 **APPLICATION USAGE**

36846 Some implementations of *iconv\_open()* use *malloc()* to allocate space for internal buffer areas.  
 36847 The *iconv\_open()* function may fail if there is insufficient storage space to accommodate these  
 36848 buffers.

36849 Conforming applications must assume that conversion descriptors are not valid after a call to  
 36850 one of the *exec* functions.36851 Application developers should consult the system documentation to determine the supported  
 36852 codesets and their naming schemes.36853 **RATIONALE**

36854 None.

36855 **FUTURE DIRECTIONS**

36856 None.

36857 **SEE ALSO**36858 *iconv()*, *iconv\_close()*36859 XBD [<fcntl.h>](#), [<iconv.h>](#)36860 **CHANGE HISTORY**

36861 First released in Issue 4. Derived from the HP-UX Manual.

36862 **Issue 7**

36863 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

36864 The *iconv\_open()* function is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**if\_freenameindex()**

System Interfaces

36865 **NAME**36866 `if_freenameindex` — free memory allocated by `if_nameindex`36867 **SYNOPSIS**36868 `#include <net/if.h>`36869 `void if_freenameindex(struct if_nameindex *ptr);`36870 **DESCRIPTION**

36871 The `if_freenameindex()` function shall free the memory allocated by `if_nameindex()`. The `ptr`  
36872 argument shall be a pointer that was returned by `if_nameindex()`. After `if_freenameindex()` has  
36873 been called, the application shall not use the array of which `ptr` is the address.

36874 **RETURN VALUE**

36875 None.

36876 **ERRORS**

36877 No errors are defined.

36878 **EXAMPLES**

36879 None.

36880 **APPLICATION USAGE**

36881 None.

36882 **RATIONALE**

36883 None.

36884 **FUTURE DIRECTIONS**

36885 None.

36886 **SEE ALSO**36887 `getsockopt()`, `if_indextoname()`, `if_nameindex()`, `if_nametoindex()`, `setsockopt()`36888 XBD `<net/if.h>`36889 **CHANGE HISTORY**

36890 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**36891 NAME**

36892 `if_indextoname` — map a network interface index to its corresponding name

**36893 SYNOPSIS**

36894 `#include <net/if.h>`

36895 `char *if_indextoname(unsigned ifindex, char *ifname);`

**36896 DESCRIPTION**

36897 The `if_indextoname()` function shall map an interface index to its corresponding name.

36898 When this function is called, *ifname* shall point to a buffer of at least {IF\_NAMESIZE} bytes. The  
36899 function shall place in this buffer the name of the interface with index *ifindex*.

**36900 RETURN VALUE**

36901 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which  
36902 points to a buffer now containing the interface name. Otherwise, the function shall return a null  
36903 pointer and set *errno* to indicate the error.

**36904 ERRORS**

36905 The `if_indextoname()` function shall fail if:

36906 [ENXIO] The interface does not exist.

**36907 EXAMPLES**

36908 None.

**36909 APPLICATION USAGE**

36910 None.

**36911 RATIONALE**

36912 None.

**36913 FUTURE DIRECTIONS**

36914 None.

**36915 SEE ALSO**

36916 `getsockopt()`, `if_freenameindex()`, `if_nameindex()`, `if_nametoindex()`, `setsockopt()`

36917 XBD `<net/if.h>`

**36918 CHANGE HISTORY**

36919 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36920 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to  
36921 {IF\_NAMESIZ} in the DESCRIPTION.

**if\_nameindex()**

System Interfaces

36922 **NAME**

36923 if\_nameindex — return all network interface names and indexes

36924 **SYNOPSIS**

```
36925 #include <net/if.h>
36926 struct if_nameindex *if_nameindex(void);
```

36927 **DESCRIPTION**

36928 The *if\_nameindex()* function shall return an array of *if\_nameindex* structures, one structure per  
 36929 interface. The end of the array is indicated by a structure with an *if\_index* field of zero and an  
 36930 *if\_name* field of NULL.

36931 Applications should call *if\_freenameindex()* to release the memory that may be dynamically  
 36932 allocated by this function, after they have finished using it.

36933 **RETURN VALUE**

36934 An array of structures identifying local interfaces. A null pointer is returned upon an error, with  
 36935 *errno* set to indicate the error.

36936 **ERRORS**

36937 The *if\_nameindex()* function may fail if:

36938 [ENOBUFS] Insufficient resources are available to complete the function.

36939 **EXAMPLES**

36940 None.

36941 **APPLICATION USAGE**

36942 None.

36943 **RATIONALE**

36944 None.

36945 **FUTURE DIRECTIONS**

36946 None.

36947 **SEE ALSO**

36948 [getsockopt\(\)](#), [if\\_freenameindex\(\)](#), [if\\_indextoname\(\)](#), [if\\_nametoindex\(\)](#), [setsockopt\(\)](#)

36949 XBD [<net/if.h>](#)

36950 **CHANGE HISTORY**

36951 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**36952 NAME**

36953 if\_nametoindex — map a network interface name to its corresponding index

**36954 SYNOPSIS**

36955 #include <net/if.h>

36956 unsigned if\_nametoindex(const char \*ifname);

**36957 DESCRIPTION**

36958 The *if\_nametoindex()* function shall return the interface index corresponding to name *ifname*.

**36959 RETURN VALUE**

36960 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

**36961 ERRORS**

36962 No errors are defined.

**36963 EXAMPLES**

36964 None.

**36965 APPLICATION USAGE**

36966 None.

**36967 RATIONALE**

36968 None.

**36969 FUTURE DIRECTIONS**

36970 None.

**36971 SEE ALSO**

36972 [getsockopt\(\)](#), [if\\_freenameindex\(\)](#), [if\\_indextoname\(\)](#), [if\\_nameindex\(\)](#), [setsockopt\(\)](#)

36973 XBD [<net/if.h>](#)

**36974 CHANGE HISTORY**

36975 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**ilogb()**36976 **NAME**36977 `ilogb, ilogbf, ilogbl` — return an unbiased exponent36978 **SYNOPSIS**

```
36979 #include <math.h>
36980 int ilogb(double x);
36981 int ilogbf(float x);
36982 int ilogbl(long double x);
```

36983 **DESCRIPTION**

36984 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36985 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36986 volume of POSIX.1-2008 defers to the ISO C standard.

36987 These functions shall return the exponent part of their argument  $x$ . Formally, the return value is  
 36988 the integral part of  $\log_r |x|$  as a signed integral value, for non-zero  $x$ , where  $r$  is the radix of the  
 36989 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

36990 An application wishing to check for error situations should set `errno` to zero and call  
 36991 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 36992 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 36993 zero, an error has occurred.

36994 **RETURN VALUE**

36995 Upon successful completion, these functions shall return the exponent part of  $x$  as a signed  
 36996 integer value. They are equivalent to calling the corresponding `logb()` function and casting the  
 36997 returned value to type `int`.

36998 XSI If  $x$  is 0, the value `FP_ILOGB0` shall be returned. On XSI-conformant systems, a domain error  
 36999 shall occur;

37000 CX otherwise, a domain error may occur.

37001 XSI If  $x$  is  $\pm\text{Inf}$ , the value `{INT_MAX}` shall be returned. On XSI-conformant systems, a domain  
 37002 error shall occur;

37003 CX otherwise, a domain error may occur.

37004 XSI If  $x$  is a NaN, the value `FP_ILOGBNAN` shall be returned. On XSI-conformant systems, a  
 37005 domain error shall occur;

37006 CX otherwise, a domain error may occur.

37007 MX If the correct value is greater than `{INT_MAX}`, a domain error shall occur and an unspecified  
 37008 XSI value shall be returned. On XSI-conformant systems, a domain error shall occur and  
 37009 `{INT_MAX}` shall be returned.

37010 MX If the correct value is less than `{INT_MIN}`, a domain error shall occur and an unspecified value  
 37011 XSI shall be returned. On XSI-conformant systems, a domain error shall occur and `{INT_MIN}` shall  
 37012 be returned.

37013 **ERRORS**

37014 These functions shall fail if:

37015 XSI | MX **Domain Error** The correct value is not representable as an integer.

37016 XSI The  $x$  argument is zero, NaN, or  $\pm\text{Inf}$ .

37017 XSI | MX If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 37018 then `errno` shall be set to `[EDOM]`. If the integer expression `(math_errhandling  
 37019 & MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception  
 37020 shall be raised.

37021 These functions may fail if:

37022 Domain Error The  $x$  argument is zero, NaN, or  $\pm\text{Inf}$ .

37023 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
37024 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling* &  
37025 MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
37026 shall be raised.

#### 37027 EXAMPLES

37028 None.

#### 37029 APPLICATION USAGE

37030 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
37031 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 37032 RATIONALE

37033 The errors come from taking the expected floating-point value and converting it to **int**, which is  
37034 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not  
37035 representable in a type **int**), so should be a domain error.

37036 There are no known implementations that overflow. For overflow to happen, {INT\_MAX} must  
37037 be less than  $\text{LDBL\_MAX\_EXP} \cdot \log_2(\text{FLT\_RADIX})$  or {INT\_MIN} must be greater than  
37038  $\text{LDBL\_MIN\_EXP} \cdot \log_2(\text{FLT\_RADIX})$  if subnormals are not supported, or {INT\_MIN} must be  
37039 greater than  $(\text{LDBL\_MIN\_EXP} - \text{LDBL\_MANT\_DIG}) \cdot \log_2(\text{FLT\_RADIX})$  if subnormals are  
37040 supported.

#### 37041 FUTURE DIRECTIONS

37042 None.

#### 37043 SEE ALSO

37044 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [logb\(\)](#), [scalbln\(\)](#)

37045 XBD Section 4.19 (on page 116), [<float.h>](#), [<math.h>](#)

#### 37046 CHANGE HISTORY

37047 First released in Issue 4, Version 2.

#### 37048 Issue 5

37049 Moved from X/OPEN UNIX extension to BASE.

#### 37050 Issue 6

37051 The *ilogb()* function is no longer marked as an extension.

37052 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999  
37053 standard.

37054 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

37055 Functionality relating to the XSI option is marked.

#### 37056 Issue 7

37057 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79  
37058 (SD5-XSH-ERN-72) are applied.

**imaxabs()**

System Interfaces

37059 **NAME**37060 `imaxabs` — return absolute value37061 **SYNOPSIS**37062 `#include <inttypes.h>`37063 `intmax_t imaxabs(intmax_t j);`37064 **DESCRIPTION**37065 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
37066 conflict between the requirements described here and the ISO C standard is unintentional. This  
37067 volume of POSIX.1-2008 defers to the ISO C standard.37068 The `imaxabs()` function shall compute the absolute value of an integer *j*. If the result cannot be  
37069 represented, the behavior is undefined.37070 **RETURN VALUE**37071 The `imaxabs()` function shall return the absolute value.37072 **ERRORS**

37073 No errors are defined.

37074 **EXAMPLES**

37075 None.

37076 **APPLICATION USAGE**

37077 The absolute value of the most negative number cannot be represented in two's complement.

37078 **RATIONALE**

37079 None.

37080 **FUTURE DIRECTIONS**

37081 None.

37082 **SEE ALSO**37083 [imaxdiv\(\)](#)37084 XBD [<inttypes.h>](#)37085 **CHANGE HISTORY**

37086 First released in Issue 6 Derived from the ISO/IEC 9899:1999 standard.

37087 **NAME**37088 `imaxdiv` — return quotient and remainder37089 **SYNOPSIS**37090 `#include <inttypes.h>`37091 `imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`37092 **DESCRIPTION**

37093 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37094 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37095 volume of POSIX.1-2008 defers to the ISO C standard.

37096 The `imaxdiv()` function shall compute `numer / denom` and `numer % denom` in a single operation.37097 **RETURN VALUE**

37098 The `imaxdiv()` function shall return a structure of type `imaxdiv_t`, comprising both the quotient  
 37099 and the remainder. The structure shall contain (in either order) the members `quot` (the quotient)  
 37100 and `rem` (the remainder), each of which has type `intmax_t`.

37101 If either part of the result cannot be represented, the behavior is undefined.

37102 **ERRORS**

37103 No errors are defined.

37104 **EXAMPLES**

37105 None.

37106 **APPLICATION USAGE**

37107 None.

37108 **RATIONALE**

37109 None.

37110 **FUTURE DIRECTIONS**

37111 None.

37112 **SEE ALSO**37113 `imaxabs()`37114 XBD `<inttypes.h>`37115 **CHANGE HISTORY**

37116 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**inet\_addr()**37117 **NAME**37118 `inet_addr, inet_ntoa` — IPv4 address manipulation37119 **SYNOPSIS**

```
37120 #include <arpa/inet.h>
37121 in_addr_t inet_addr(const char *cp);
37122 char *inet_ntoa(struct in_addr in);
```

37123 **DESCRIPTION**

37124 The `inet_addr()` function shall convert the string pointed to by `cp`, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

37126 The `inet_ntoa()` function shall convert the Internet host address specified by `in` to a string in the Internet standard dot notation.

37128 The `inet_ntoa()` function need not be thread-safe.

37129 All Internet addresses shall be returned in network order (bytes ordered from left to right).

37130 Values specified using IPv4 dotted decimal notation take one of the following forms:

37131 `a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and  
37132 assigned, from left to right, to the four bytes of an Internet address.

37133 `a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit  
37134 quantity and placed in the rightmost two bytes of the network address. This makes  
37135 the three-part address format convenient for specifying Class B network addresses  
37136 as "128.net.host".

37137 `a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit  
37138 quantity and placed in the rightmost three bytes of the network address. This  
37139 makes the two-part address format convenient for specifying Class A network  
37140 addresses as "net.host".

37141 `a` When only one part is given, the value shall be stored directly in the network  
37142 address without any byte rearrangement.

37143 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or  
37144 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;  
37145 otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).

37146 **RETURN VALUE**

37147 Upon successful completion, `inet_addr()` shall return the Internet address. Otherwise, it shall  
37148 return `(in_addr_t)(-1)`.

37149 The `inet_ntoa()` function shall return a pointer to the network address in Internet standard dot  
37150 notation.

37151 **ERRORS**

37152 No errors are defined.

37153 **EXAMPLES**

37154 None.

37155 **APPLICATION USAGE**37156 The return value of *inet\_ntoa()* may point to static data that may be overwritten by subsequent  
37157 calls to *inet\_ntoa()*.37158 **RATIONALE**

37159 None.

37160 **FUTURE DIRECTIONS**

37161 None.

37162 **SEE ALSO**37163 *endhostent()*, *endnetent()*37164 XBD <[arpa/inet.h](#)>37165 **CHANGE HISTORY**

37166 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37167 **Issue 7**

37168 Austin Group Interpretation 1003.1-2001 #156 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**inet\_ntop()**37169 **NAME**37170 `inet_ntop, inet_pton` — convert IPv4 and IPv6 addresses between binary and text form37171 **SYNOPSIS**

```
37172 #include <arpa/inet.h>
37173
37174 const char *inet_ntop(int af, const void *restrict src,
37175 char *restrict dst, socklen_t size);
37176
37177 int inet_pton(int af, const char *restrict src, void *restrict dst);
```

37176 **DESCRIPTION**

37177 The `inet_ntop()` function shall convert a numeric address into a text string suitable for  
 37178 IP6 presentation. The `af` argument shall specify the family of the address. This can be `AF_INET` or  
 37179 `AF_INET6`. The `src` argument points to a buffer holding an IPv4 address if the `af` argument is  
 37180 IP6 `AF_INET`, or an IPv6 address if the `af` argument is `AF_INET6`; the address must be in network  
 37181 byte order. The `dst` argument points to a buffer where the function stores the resulting text string;  
 37182 it shall not be NULL. The `size` argument specifies the size of this buffer, which shall be large  
 37183 IP6 enough to hold the text string (`INET_ADDRSTRLEN` characters for IPv4,  
 37184 `INET6_ADDRSTRLEN` characters for IPv6).

37185 The `inet_pton()` function shall convert an address in its standard text presentation form into its  
 37186 IP6 numeric binary form. The `af` argument shall specify the family of the address. The `AF_INET` and  
 37187 `AF_INET6` address families shall be supported. The `src` argument points to the string being  
 37188 passed in. The `dst` argument points to a buffer into which the function stores the numeric  
 37189 IP6 address; this shall be large enough to hold the numeric address (32 bits for `AF_INET`, 128 bits  
 37190 for `AF_INET6`).

37191 If the `af` argument of `inet_pton()` is `AF_INET`, the `src` string shall be in the standard IPv4 dotted-  
 37192 decimal form:

37193 `ddd.ddd.ddd.ddd`

37194 where "ddd" is a one to three digit decimal number between 0 and 255 (see `inet_addr()`). The  
 37195 `inet_pton()` function does not accept other formats (such as the octal numbers, hexadecimal  
 37196 numbers, and fewer than four numbers that `inet_addr()` accepts).

37197 IP6 If the `af` argument of `inet_pton()` is `AF_INET6`, the `src` string shall be in one of the following  
 37198 standard IPv6 text forms:

- 37199 1. The preferred form is "`x:x:x:x:x:x:x:x`", where the 'x's are the hexadecimal values  
 37200 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be  
 37201 omitted, but there shall be at least one numeral in every field.
- 37202 2. A string of contiguous zero fields in the preferred form can be shown as "`::`". The "`::`"  
 37203 can only appear once in an address. Unspecified addresses ("`0:0:0:0:0:0:0:0`") may  
 37204 be represented simply as "`::`".
- 37205 3. A third form that is sometimes more convenient when dealing with a mixed environment  
 37206 of IPv4 and IPv6 nodes is "`x:x:x:x:x:x.d.d.d.d`", where the 'x's are the  
 37207 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the  
 37208 decimal values of the four low-order 8-bit pieces of the address (standard IPv4  
 37209 representation).

37210 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in  
 37211 RFC 2373.

37212 **RETURN VALUE**

37213 The *inet\_ntop()* function shall return a pointer to the buffer containing the text string if the  
 37214 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

37215 The *inet\_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by  
 37216 IP6 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string  
 37217 or a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is  
 37218 unknown.

37219 **ERRORS**

37220 The *inet\_ntop()* and *inet\_pton()* functions shall fail if:

37221 [EAFNOSUPPORT]

37222 The *af* argument is invalid.

37223 [ENOSPC] The size of the *inet\_ntop()* result buffer is inadequate.

37224 **EXAMPLES**

37225 None.

37226 **APPLICATION USAGE**

37227 None.

37228 **RATIONALE**

37229 None.

37230 **FUTURE DIRECTIONS**

37231 None.

37232 **SEE ALSO**

37233 XBD [<arpa/inet.h>](#)

37234 **CHANGE HISTORY**

37235 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37236 IPv6 extensions are marked.

37237 The **restrict** keyword is added to the *inet\_ntop()* and *inet\_pton()* prototypes for alignment with  
 37238 the ISO/IEC 9899:1999 standard.

37239 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding "the address must  
 37240 be in network byte order" to the end of the fourth sentence of the first paragraph in the  
 37241 DESCRIPTION.

**initstate()**37242 **NAME**37243 `initstate`, `random`, `setstate`, `srandom` — pseudo-random number functions37244 **SYNOPSIS**

```

37245 XSI #include <stdlib.h>
37246 char *initstate(unsigned seed, char *state, size_t size);
37247 long random(void);
37248 char *setstate(char *state);
37249 void srandom(unsigned seed);

```

37250 **DESCRIPTION**

37251 The `random()` function shall use a non-linear additive feedback random-number generator  
 37252 employing a default state array size of 31 **long** integers to return successive pseudo-random  
 37253 numbers in the range from 0 to  $2^{31}-1$ . The period of this random-number generator is  
 37254 approximately  $16 \times (2^{31}-1)$ . The size of the state array determines the period of the random-  
 37255 number generator. Increasing the state array size shall increase the period.

37256 With 256 bytes of state information, the period of the random-number generator shall be greater  
 37257 than  $2^{69}$ .

37258 Like `rand()`, `random()` shall produce by default a sequence of numbers that can be duplicated by  
 37259 calling `srandom()` with 1 as the seed.

37260 The `srandom()` function shall initialize the current state array using the value of `seed`.

37261 The `initstate()` and `setstate()` functions handle restarting and changing random-number  
 37262 generators. The `initstate()` function allows a state array, pointed to by the `state` argument, to be  
 37263 initialized for future use. The `size` argument, which specifies the size in bytes of the state array,  
 37264 shall be used by `initstate()` to decide what type of random-number generator to use; the larger  
 37265 the state array, the more random the numbers. Values for the amount of state information are 8,  
 37266 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one  
 37267 of these values. If `initstate()` is called with  $8 \leq \text{size} < 32$ , then `random()` shall use a simple linear  
 37268 congruential random number generator. The `seed` argument specifies a starting point for the  
 37269 random-number sequence and provides for restarting at the same point. The `initstate()` function  
 37270 shall return a pointer to the previous state information array.

37271 If `initstate()` has not been called, then `random()` shall behave as though `initstate()` had been called  
 37272 with `seed=1` and `size=128`.

37273 Once a state has been initialized, `setstate()` allows switching between state arrays. The array  
 37274 defined by the `state` argument shall be used for further random-number generation until  
 37275 `initstate()` is called or `setstate()` is called again. The `setstate()` function shall return a pointer to the  
 37276 previous state array.

37277 **RETURN VALUE**

37278 If `initstate()` is called with `size` less than 8, it shall return NULL.

37279 The `random()` function shall return the generated pseudo-random number.

37280 The `srandom()` function shall not return a value.

37281 Upon successful completion, `initstate()` and `setstate()` shall return a pointer to the previous state  
 37282 array; otherwise, a null pointer shall be returned.

37283 **ERRORS**

37284 No errors are defined.

37285 **EXAMPLES**

37286 None.

37287 **APPLICATION USAGE**

37288 After initialization, a state array can be restarted at a different point in one of two ways:

- 37289 1. The *initstate()* function can be used, with the desired seed, state array, and size of the  
37290 array.
- 37291 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with  
37292 the desired seed. The advantage of using both of these functions is that the size of the  
37293 state array does not have to be saved once it is initialized.

37294 Although some implementations of *random()* have written messages to standard error, such  
37295 implementations do not conform to POSIX.1-2008.

37296 Issue 5 restored the historical behavior of this function.

37297 Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when  
37298 an independent random number sequence in multiple threads is required.

37299 **RATIONALE**

37300 None.

37301 **FUTURE DIRECTIONS**

37302 None.

37303 **SEE ALSO**37304 *drand48()*, *rand()*37305 XBD `<stdlib.h>`37306 **CHANGE HISTORY**

37307 First released in Issue 4, Version 2.

37308 **Issue 5**

37309 Moved from X/OPEN UNIX extension to BASE.

37310 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than  
37311 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph  
37312 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE  
37313 indicating that these changes restore the historical behavior of the function.

37314 **Issue 6**

37315 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

37316 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand\_r()* from the  
37317 list of suggested functions in the APPLICATION USAGE section.

37318 **Issue 7**37319 The type of the first argument to *setstate()* is changed from **const char \*** to **char \***.

**insque()**37320 **NAME**37321 `insque, remque` — insert or remove an element in a queue37322 **SYNOPSIS**

```

37323 XSI      #include <search.h>
37324         void insque(void *element, void *pred);
37325         void remque(void *element);

```

37326 **DESCRIPTION**

37327 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The  
 37328 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it  
 37329 defines a structure in which the first two members of the structure are pointers to the same type  
 37330 of structure, and any further members are application-specific. The first member of the structure  
 37331 is a forward pointer to the next entry in the queue. The second member is a backward pointer to  
 37332 the previous entry in the queue. If the queue is linear, the queue is terminated with null  
 37333 pointers. The names of the structure and of the pointer members are not subject to any special  
 37334 restriction.

37335 The *insque()* function shall insert the element pointed to by *element* into a queue immediately  
 37336 after the element pointed to by *pred*.

37337 The *remque()* function shall remove the element pointed to by *element* from a queue.

37338 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the  
 37339 initial element of the queue, shall initialize the forward and backward pointers of *element* to null  
 37340 pointers.

37341 If the queue is to be used as a circular list, the application shall ensure it initializes the forward  
 37342 pointer and the backward pointer of the initial element of the queue to the element's own  
 37343 address.

37344 **RETURN VALUE**37345 The *insque()* and *remque()* functions do not return a value.37346 **ERRORS**

37347 No errors are defined.

37348 **EXAMPLES**37349 **Creating a Linear Linked List**

37350 The following example creates a linear linked list.

```

37351 #include <search.h>
37352 .
37353 struct myque element1;
37354 struct myque element2;
37355
37356 char *data1 = "DATA1";
37357 char *data2 = "DATA2";
37358 ...
37359 element1.data = data1;
37360 element2.data = data2;
37361
37362 insque (&element1, NULL);
37363 insque (&element2, &element1);

```

37362 **Creating a Circular Linked List**

37363 The following example creates a circular linked list.

```

37364 #include <search.h>
37365 ...
37366 struct myque element1;
37367 struct myque element2;

37368 char *data1 = "DATA1";
37369 char *data2 = "DATA2";
37370 ...
37371 element1.data = data1;
37372 element2.data = data2;

37373 element1.fwd = &element1;
37374 element1.bck = &element1;

37375 insque (&element2, &element1);

```

37376 **Removing an Element**37377 The following example removes the element pointed to by *element1*.

```

37378 #include <search.h>
37379 ...
37380 struct myque element1;
37381 ...
37382 remove (&element1);

```

37383 **APPLICATION USAGE**

37384 The historical implementations of these functions described the arguments as being of type  
 37385 **struct qelem \*** rather than as being of type **void \*** as defined here. In those implementations,  
 37386 **struct qelem** was commonly defined in **<search.h>** as:

```

37387 struct qelem {
37388     struct qelem *q_forw;
37389     struct qelem *q_back;
37390 };

```

37391 Applications using these functions, however, were never able to use this structure directly since  
 37392 it provided no room for the actual data contained in the elements. Most applications defined  
 37393 structures that contained the two pointers as the initial elements and also provided space for,  
 37394 pointers to, the object's data. Applications that used these functions to update more than one  
 37395 type of table also had the problem of specifying two or more different structures with the same  
 37396 name, if they literally used **struct qelem** as specified.

37397 As described here, the implementations were actually expecting a structure type where the first  
 37398 two members were forward and backward pointers to structures. With C compilers that didn't  
 37399 provide function prototypes, applications used structures as specified in the DESCRIPTION  
 37400 above and the compiler did what the application expected.

37401 If this method had been carried forward with an ISO C standard compiler and the historical  
 37402 function prototype, most applications would have to be modified to cast pointers to the  
 37403 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By  
 37404 specifying **void \*** as the argument type, applications do not need to change (unless they  
 37405 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

**insque()**37406 **RATIONALE**

37407 None.

37408 **FUTURE DIRECTIONS**

37409 None.

37410 **SEE ALSO**37411 XBD [<search.h>](#)37412 **CHANGE HISTORY**

37413 First released in Issue 4, Version 2.

37414 **Issue 5**

37415 Moved from X/OPEN UNIX extension to BASE.

37416 **Issue 6**

37417 The normative text is updated to avoid use of the term “must” for application requirements.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

37418 **NAME**37419 ioctl — control a STREAMS device (**STREAMS**)37420 **SYNOPSIS**

```
37421 OB XSR #include <stropts.h>
37422 int ioctl(int fildes, int request, ... /* arg */);
```

37423 **DESCRIPTION**

37424 The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-  
 37425 STREAMS devices, the functions performed by this call are unspecified. The *request* argument  
 37426 and an optional third argument (with varying type) shall be passed to and interpreted by the  
 37427 appropriate part of the STREAM associated with *fildes*.

37428 The *fildes* argument is an open file descriptor that refers to a device.

37429 The *request* argument selects the control function to be performed and shall depend on the  
 37430 STREAMS device being addressed.

37431 The *arg* argument represents additional information that is needed by this specific STREAMS  
 37432 device to perform the requested function. The type of *arg* depends upon the particular control  
 37433 request, but it shall be either an integer or a pointer to a device-specific data structure.

37434 The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply  
 37435 to each individual command are described below.

37436 The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS  
 37437 files:

37438 **I\_PUSH** Pushes the module whose name is pointed to by *arg* onto the top of the current  
 37439 STREAM, just below the STREAM head. It then calls the *open()* function of the  
 37440 newly-pushed module.

37441 The *ioctl()* function with the **I\_PUSH** command shall fail if:

37442 [EINVAL] Invalid module name.

37443 [ENXIO] Open function of new module failed.

37444 [ENXIO] Hangup received on *fildes*.

37445 **I\_POP** Removes the module just below the STREAM head of the STREAM pointed to  
 37446 by *fildes*. The *arg* argument should be 0 in an **I\_POP** request.

37447 The *ioctl()* function with the **I\_POP** command shall fail if:

37448 [EINVAL] No module present in the STREAM.

37449 [ENXIO] Hangup received on *fildes*.

37450 **I\_LOOK** Retrieves the name of the module just below the STREAM head of the  
 37451 STREAM pointed to by *fildes*, and places it in a character string pointed to by  
 37452 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,  
 37453 where FMNAMESZ is defined in **<stropts.h>**.

37454 The *ioctl()* function with the **I\_LOOK** command shall fail if:

37455 [EINVAL] No module present in the STREAM.

37456 **I\_FLUSH** Flushes read and/or write queues, depending on the value of *arg*. Valid *arg*  
 37457 values are:

**ioctl()**

37458	FLUSHR	Flush all read queues.
37459	FLUSHW	Flush all write queues.
37460	FLUSHRW	Flush all read and all write queues.
37461	The <i>ioctl()</i> function with the <code>I_FLUSH</code> command shall fail if:	
37462	[EINVAL]	Invalid <i>arg</i> value.
37463	[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
37464		
37465	[ENXIO]	Hangup received on <i>fildev</i> .
37466	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a <b>bandinfo</b> structure. The <i>bi_flag</i> member may be one of <code>FLUSHR</code> , <code>FLUSHW</code> , or <code>FLUSHRW</code> as described above. The <i>bi_pri</i> member determines the priority band to be flushed.
37467		
37468		
37469		
37470	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . <code>I_SETSIG</code> supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:
37471		
37472		
37473		
37474		
37475		
37476	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37477		
37478		
37479	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37480		
37481		
37482	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37483		
37484		
37485	S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37486		
37487		
37488	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
37489		
37490		
37491		
37492	S_WRNORM	Equivalent to <code>S_OUTPUT</code> .
37493	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
37494		
37495		
37496		
37497	S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
37498		
37499		

37500		S_ERROR	Notification of an error condition has reached the STREAM head.
37501			
37502		S_HANGUP	Notification of a hangup has reached the STREAM head.
37503		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
37504			
37505			
37506			If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .
37507			
37508			Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
37509			
37510			
37511			
37512			The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
37513		[EINVAL]	The value of <i>arg</i> is invalid.
37514		[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
37515			
37516		[EAGAIN]	There were insufficient resources to store the signal request.
37517	I_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an <b>int</b> pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
37518			
37519			
37520			
37521			The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
37522		[EINVAL]	Process is not registered to receive the SIGPOLL signal.
37523	I_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.
37524			
37525			
37526			The <i>ioctl()</i> function with the I_FIND command shall fail if:
37527		[EINVAL]	<i>arg</i> does not contain a valid module name.
37528	I_PEEK		Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a <b>strpeek</b> structure.
37529			
37530			
37531			
37532			The application shall ensure that the <i>maxlen</i> member in the <b>ctlbuf</b> and <b>datbuf</b> structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.
37533			
37534			
37535			
37536			
37537			
37538			I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, <b>ctlbuf</b> specifies information in the control buffer, <b>datbuf</b> specifies information in the data
37539			
37540			
37541			
37542			

**ioctl()**

37543		buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
37544	I_SRDOPT	Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:
37545		
37546	RNORM	Byte-stream mode, the default.
37547	RMSGD	Message-discard mode.
37548	RMSGN	Message-nondiscard mode.
37549		The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.
37550		
37551		
37552		In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in <i>arg</i> :
37553		
37554	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
37555		
37556	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
37557		
37558	RPROTDIS	Discard the control part of a message, delivering any data portion, when a process issues a <i>read()</i> .
37559		
37560		The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:
37561	[EINVAL]	The <i>arg</i> argument is not valid.
37562	I_GRDOPT	Returns the current read mode setting, as described above, in an <b>int</b> pointed to by the argument <i>arg</i> . Read modes are described in <i>read()</i> .
37563		
37564	I_NREAD	Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the <b>int</b> pointed to by <i>arg</i> . The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i> return value is greater than 0, this indicates that a zero-length message is next on the queue.
37565		
37566		
37567		
37568		
37569		
37570	I_FDINSERT	Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The <i>arg</i> argument points to a <b>strfdinsert</b> structure.
37571		
37572		
37573		
37574		
37575		The application shall ensure that the <i>len</i> member in the <b>ctlbuf strbuf</b> structure is set to the size of a <b>t_uscalar_t</b> plus the number of bytes of control information to be sent with the message. The <i>fildev</i> member specifies the file descriptor of the other STREAM, and the <i>offset</i> member, which must be suitably aligned for use as a <b>t_uscalar_t</b> , specifies the offset from the start of the control buffer where I_FDINSERT shall store a <b>t_uscalar_t</b> whose interpretation is specific to the STREAM end. The application shall ensure that the <i>len</i> member in the <b>databuf strbuf</b> structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.
37576		
37577		
37578		
37579		
37580		
37581		
37582		
37583		
37584		
37585		The <i>flags</i> member specifies the type of message to be created. A normal message is created if <i>flags</i> is set to 0, and a high-priority message is created if
37586		

37587 37588 37589 37590 37591		<i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NONBLOCK is set. Instead, it fails and sets <i>errno</i> to [EAGAIN].
37592 37593 37594 37595		I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message is sent.
37596		The <i>ioctl()</i> function with the I_FDINSERT command shall fail if:
37597 37598 37599	[EAGAIN]	A non-priority message is specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions.
37600	[EAGAIN] or [ENOSR]	
37601 37602		Buffers cannot be allocated for the message that is to be created.
37603	[EINVAL]	One of the following:
37604 37605		— The <i>fildev</i> member of the <b>strfdinsert</b> structure is not a valid, open STREAM file descriptor.
37606 37607		— The size of a <b>t_uscalar_t</b> plus <i>offset</i> is greater than the <i>len</i> member for the buffer specified through <b>ctlbuf</b> .
37608 37609		— The <i>offset</i> member does not specify a properly-aligned location in the data buffer.
37610		— An undefined value is stored in <i>flags</i> .
37611 37612 37613	[ENXIO]	Hangup received on the STREAM identified by either the <i>fildev</i> argument or the <i>fildev</i> member of the <b>strfdinsert</b> structure.
37614 37615 37616 37617 37618 37619 37620 37621	[ERANGE]	The <i>len</i> member for the buffer specified through <b>databuf</b> does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module; or the <i>len</i> member for the buffer specified through <b>databuf</b> is larger than the maximum configured size of the data part of a message; or the <i>len</i> member for the buffer specified through <b>ctlbuf</b> is larger than the maximum configured size of the control part of a message.
37622 37623	I_STR	Constructs an internal STREAMS <i>ioctl()</i> message from the data pointed to by <i>arg</i> , and sends that message downstream.
37624 37625 37626 37627 37628 37629		This mechanism is provided to send <i>ioctl()</i> requests to downstream modules and drivers. It allows information to be sent with <i>ioctl()</i> , and returns to the process any information sent upstream by the downstream recipient. I_STR shall block until the system responds with either a positive or negative acknowledgement message, or until the request times out after some period of time. If the request times out, it shall fail with <i>errno</i> set to [ETIME].
37630 37631		At most, one I_STR can be active on a STREAM. Further I_STR calls shall block until the active I_STR completes at the STREAM head. The default

**ioctl()**

37632		timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
37633		no effect on this call.
37634		To send requests downstream, the application shall ensure that <i>arg</i> points to a
37635		<b>strioc</b> structure.
37636		The <i>ic_cmd</i> member is the internal <i>ioctl()</i> command intended for a
37637		downstream module or driver and <i>ic_timeout</i> is the number of seconds
37638		(-1=infinite, 0=use implementation-defined timeout interval, >0=as specified)
37639		an I_STR request shall wait for acknowledgement before timing out. <i>ic_len</i> is
37640		the number of bytes in the data argument, and <i>ic_dp</i> is a pointer to the data
37641		argument. The <i>ic_len</i> member has two uses: on input, it contains the length of
37642		the data argument passed in, and on return from the command, it contains the
37643		number of bytes being returned to the process (the buffer pointed to by <i>ic_dp</i>
37644		should be large enough to contain the maximum amount of data that any
37645		module or the driver in the STREAM can return).
37646		The STREAM head shall convert the information pointed to by the <b>strioc</b>
37647		structure to an internal <i>ioctl()</i> command message and send it downstream.
37648		The <i>ioctl()</i> function with the I_STR command shall fail if:
37649		[EAGAIN] or [ENOSR]
37650		Unable to allocate buffers for the <i>ioctl()</i> message.
37651		[EINVAL] The <i>ic_len</i> member is less than 0 or larger than the
37652		maximum configured size of the data part of a message, or
37653		<i>ic_timeout</i> is less than -1.
37654		[ENXIO] Hangup received on <i>fil</i> .
37655		[ETIME] A downstream <i>ioctl()</i> timed out before acknowledgement
37656		was received.
37657		An I_STR can also fail while waiting for an acknowledgement if a message
37658		indicating an error or a hangup is received at the STREAM head. In addition,
37659		an error code can be returned in the positive or negative acknowledgement
37660		message, in the event the <i>ioctl()</i> command sent downstream fails. For these
37661		cases, I_STR shall fail with <i>errno</i> set to the value in the message.
37662	I_SWROPT	Sets the write mode using the value of the argument <i>arg</i> . Valid bit settings for
37663		<i>arg</i> are:
37664		SNDZERO Send a zero-length message downstream when a <i>write()</i> of 0
37665		bytes occurs. To not send a zero-length message when a
37666		<i>write()</i> of 0 bytes occurs, the application shall ensure that
37667		this bit is not set in <i>arg</i> (for example, <i>arg</i> would be set to 0).
37668		The <i>ioctl()</i> function with the I_SWROPT command shall fail if:
37669		[EINVAL] <i>arg</i> is not the above value.
37670	I_GWROPT	Returns the current write mode setting, as described above, in the <b>int</b> that is
37671		pointed to by the argument <i>arg</i> .
37672	I_SENDFD	Creates a new reference to the open file description associated with the file
37673		descriptor <i>arg</i> , and writes a message on the STREAMS-based pipe <i>fil</i>
37674		containing this reference, together with the user ID and group ID of the calling
37675		process.

37676		The <i>ioctl()</i> function with the <code>L_SENDFD</code> command shall fail if:
37677	[EAGAIN]	The sending STREAM is unable to allocate a message block to contain the file pointer; or the read queue of the receiving STREAM head is full and cannot accept the message sent by <code>L_SENDFD</code> .
37678		
37679		
37680		
37681	[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37682	[EINVAL]	The <i>fildev</i> argument is not connected to a STREAM pipe.
37683	[ENXIO]	Hangup received on <i>fildev</i> .
37684		The <i>ioctl()</i> function with the <code>L_SENDFD</code> command may fail if:
37685	[EINVAL]	The <i>arg</i> argument is equal to the <i>fildev</i> argument.
37686	<code>L_RECVFD</code>	Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the <code>L_SENDFD</code> command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a <code>strrecvfd</code> data structure as defined in <code>&lt;stropts.h&gt;</code> .
37687		
37688		
37689		
37690		
37691		The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.
37692		
37693		If <code>O_NONBLOCK</code> is not set, <code>L_RECVFD</code> shall block until a message is present at the STREAM head. If <code>O_NONBLOCK</code> is set, <code>L_RECVFD</code> shall fail with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.
37694		
37695		
37696		If the message at the STREAM head is a message sent by an <code>L_SENDFD</code> , a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the <code>strrecvfd</code> structure pointed to by <i>arg</i> .
37697		
37698		
37699		
37700		The <i>ioctl()</i> function with the <code>L_RECVFD</code> command shall fail if:
37701	[EAGAIN]	A message is not present at the STREAM head read queue and the <code>O_NONBLOCK</code> flag is set.
37702		
37703	[EBADMSG]	The message at the STREAM head read queue is not a message containing a passed file descriptor.
37704		
37705	[EMFILE]	All file descriptors available to the process are currently open.
37706		
37707	[ENXIO]	Hangup received on <i>fildev</i> .
37708	<code>L_LIST</code>	Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a <code>str_list</code> structure.
37709		
37710		
37711		
37712		
37713		The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the <code>str_list</code> structure shall contain the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the
37714		
37715		
37716		
37717		
37718		

**ioctl()**

37719		STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules ( <i>sl_nmods</i> ) is satisfied.
37720		
37721		The <i>ioctl()</i> function with the <code>L_LIST</code> command shall fail if:
37722		[EINVAL]       The <i>sl_nmods</i> member is less than 1.
37723		[EAGAIN] or [ENOSR]
37724		Unable to allocate buffers.
37725	<code>I_ATMARK</code>	Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:
37726		
37727		
37728		
37729		<code>ANYMARK</code> Check if the message is marked.
37730		<code>LASTMARK</code> Check if the message is the last one marked on the queue.
37731		The bitwise-inclusive OR of the flags <code>ANYMARK</code> and <code>LASTMARK</code> is permitted.
37732		
37733		The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.
37734		
37735		The <i>ioctl()</i> function with the <code>I_ATMARK</code> command shall fail if:
37736		[EINVAL]       Invalid <i>arg</i> value.
37737	<code>I_CKBAND</code>	Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or -1 on error. <i>arg</i> should be of type <code>int</code> .
37738		
37739		
37740		The <i>ioctl()</i> function with the <code>I_CKBAND</code> command shall fail if:
37741		[EINVAL]       Invalid <i>arg</i> value.
37742	<code>I_GETBAND</code>	Returns the priority band of the first message on the STREAM head read queue in the integer referenced by <i>arg</i> .
37743		
37744		The <i>ioctl()</i> function with the <code>I_GETBAND</code> command shall fail if:
37745		[ENODATA]     No message on the STREAM head read queue.
37746	<code>I_CANPUT</code>	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or -1 on error.
37747		
37748		
37749		The <i>ioctl()</i> function with the <code>I_CANPUT</code> command shall fail if:
37750		[EINVAL]       Invalid <i>arg</i> value.
37751	<code>I_SETCLTIME</code>	This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If <code>I_SETCLTIME</code> is not performed on a STREAM, an implementation-defined default timeout interval is used.
37752		
37753		
37754		
37755		
37756		
37757		
37758		

37759 The *ioctl()* function with the `I_SETCLTIME` command shall fail if:

37760 [EINVAL] Invalid *arg* value.

37761 `I_GETCLTIME` Returns the close time delay in the integer pointed to by *arg*.

### 37762 Multiplexed STREAMS Configurations

37763 The following commands are used for connecting and disconnecting multiplexed STREAMS  
37764 configurations. These commands use an implementation-defined default timeout interval.

37765 `I_LINK` Connects two STREAMS, where *fildev* is the file descriptor of the STREAM  
37766 connected to the multiplexing driver, and *arg* is the file descriptor of the  
37767 STREAM connected to another driver. The STREAM designated by *arg* is  
37768 connected below the multiplexing driver. `I_LINK` requires the multiplexing  
37769 driver to send an acknowledgement message to the STREAM head regarding  
37770 the connection. This call shall return a multiplexer ID number (an identifier  
37771 used to disconnect the multiplexer; see `I_UNLINK`) on success, and `-1` on  
37772 failure.

37773 The *ioctl()* function with the `I_LINK` command shall fail if:

37774 [ENXIO] Hangup received on *fildev*.

37775 [ETIME] Timeout before acknowledgement message was received at  
37776 STREAM head.

37777 [EAGAIN] or [ENOSR]

37778 Unable to allocate STREAMS storage to perform the  
37779 `I_LINK`

37780 [EBADF] The *arg* argument is not a valid, open file descriptor.

37781 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is  
37782 not a STREAM or is already connected downstream from a  
37783 multiplexer; or the specified `I_LINK` operation would  
37784 connect the STREAM head in more than one place in the  
37785 multiplexed STREAM.

37786 An `I_LINK` can also fail while waiting for the multiplexing driver to  
37787 acknowledge the request, if a message indicating an error or a hangup is  
37788 received at the STREAM head of *fildev*. In addition, an error code can be  
37789 returned in the positive or negative acknowledgement message. For these  
37790 cases, `I_LINK` fails with *errno* set to the value in the message.

37791 `I_UNLINK` Disconnects the two STREAMS specified by *fildev* and *arg*. *fildev* is the file  
37792 descriptor of the STREAM connected to the multiplexing driver. The *arg*  
37793 argument is the multiplexer ID number that was returned by the `I_LINK`  
37794 *ioctl()* command when a STREAM was connected downstream from the  
37795 multiplexing driver. If *arg* is `MUXID_ALL`, then all STREAMS that were  
37796 connected to *fildev* shall be disconnected. As in `I_LINK`, this command requires  
37797 acknowledgement.

37798 The *ioctl()* function with the `I_UNLINK` command shall fail if:

37799 [ENXIO] Hangup received on *fildev*.

**ioctl()**

37800		[ETIME]	Timeout before acknowledgement message was received at STREAM head.
37801			
37802		[EAGAIN] or [ENOSR]	Unable to allocate buffers for the acknowledgement message.
37803			
37804			
37805		[EINVAL]	Invalid multiplexer ID number.
37806			An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of <i>fildev</i> . In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.
37807			
37808			
37809			
37810			
37811	I_PLINK		Creates a <i>persistent connection</i> between two STREAMs, where <i>fildev</i> is the file descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the file descriptor of the STREAM connected to another driver. This call shall create a persistent connection which can exist even if the file descriptor <i>fildev</i> associated with the upper STREAM to the multiplexing driver is closed. The STREAM designated by <i>arg</i> gets connected via a persistent connection below the multiplexing driver. I_PLINK requires the multiplexing driver to send an acknowledgement message to the STREAM head. This call shall return a multiplexer ID number (an identifier that may be used to disconnect the multiplexer; see I_PUNLINK) on success, and -1 on failure.
37812			
37813			
37814			
37815			
37816			
37817			
37818			
37819			
37820			
37821			The <i>ioctl()</i> function with the I_PLINK command shall fail if:
37822		[ENXIO]	Hangup received on <i>fildev</i> .
37823		[ETIME]	Timeout before acknowledgement message was received at STREAM head.
37824			
37825		[EAGAIN] or [ENOSR]	Unable to allocate STREAMS storage to perform the I_PLINK.
37826			
37827			
37828		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37829		[EINVAL]	The <i>fildev</i> argument does not support multiplexing; or <i>arg</i> is not a STREAM or is already connected downstream from a multiplexer; or the specified I_PLINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.
37830			
37831			
37832			
37833			
37834			An I_PLINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of <i>fildev</i> . In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PLINK shall fail with <i>errno</i> set to the value in the message.
37835			
37836			
37837			
37838			
37839	I_PUNLINK		Disconnects the two STREAMs specified by <i>fildev</i> and <i>arg</i> from a persistent connection. The <i>fildev</i> argument is the file descriptor of the STREAM connected to the multiplexing driver. The <i>arg</i> argument is the multiplexer ID number that was returned by the I_PLINK <i>ioctl()</i> command when a STREAM was connected downstream from the multiplexing driver. If <i>arg</i> is MUXID_ALL, then all STREAMs which are persistent connections to <i>fildev</i> shall be disconnected. As in I_PLINK, this command requires the multiplexing
37840			
37841			
37842			
37843			
37844			
37845			

- 37846 driver to acknowledge the request.
- 37847 The *ioctl()* function with the I\_PUNLINK command shall fail if:
- 37848 [ENXIO] Hangup received on *fildev*.
- 37849 [ETIME] Timeout before acknowledgement message was received at  
37850 STREAM head.
- 37851 [EAGAIN] or [ENOSR]  
37852 Unable to allocate buffers for the acknowledgement  
37853 message.
- 37854 [EINVAL] Invalid multiplexer ID number.
- 37855 An I\_PUNLINK can also fail while waiting for the multiplexing driver to  
37856 acknowledge the request if a message indicating an error or a hangup is  
37857 received at the STREAM head of *fildev*. In addition, an error code can be  
37858 returned in the positive or negative acknowledgement message. For these  
37859 cases, I\_PUNLINK shall fail with *errno* set to the value in the message.
- 37860 **RETURN VALUE**
- 37861 Upon successful completion, *ioctl()* shall return a value other than  $-1$  that depends upon the  
37862 STREAMS device control function. Otherwise, it shall return  $-1$  and set *errno* to indicate the  
37863 error.
- 37864 **ERRORS**
- 37865 Under the following general conditions, *ioctl()* shall fail if:
- 37866 [EBADF] The *fildev* argument is not a valid open file descriptor.
- 37867 [EINTR] A signal was caught during the *ioctl()* operation.
- 37868 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
37869 indirectly) downstream from a multiplexer.
- 37870 If an underlying device driver detects an error, then *ioctl()* shall fail if:
- 37871 [EINVAL] The *request* or *arg* argument is not valid for this device.
- 37872 [EIO] Some physical I/O error has occurred.
- 37873 [ENOTTY] The file associated with the *fildev* argument is not a STREAMS device that  
37874 accepts control functions.
- 37875 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service  
37876 requested cannot be performed on this particular sub-device.
- 37877 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding  
37878 device driver does not support the *ioctl()* function.
- 37879 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except  
37880 I\_UNLINK and I\_PUNLINK shall set *errno* to [EINVAL].

**ioctl()**37881 **EXAMPLES**

37882 None.

37883 **APPLICATION USAGE**

37884 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

37885 **RATIONALE**

37886 None.

37887 **FUTURE DIRECTIONS**37888 The *ioctl()* function may be removed in a future version.37889 **SEE ALSO**37890 Section 2.6 (on page 494), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*,  
37891 *sigaction()*, *write()*37892 XBD <*stropts.h*>37893 **CHANGE HISTORY**

37894 First released in Issue 4, Version 2.

37895 **Issue 5**

37896 Moved from X/OPEN UNIX extension to BASE.

37897 **Issue 6**37898 The Open Group Corrigendum U028/4 is applied, correcting text in the L\_FDINSERT [EINVAL]  
37899 case to refer to *ctlbuf*.

37900 This function is marked as part of the XSI STREAMS Option Group.

37901 The normative text is updated to avoid use of the term “must” for application requirements.

37902 **Issue 7**37903 Austin Group Interpretation 1003.1-2001 #155 is applied, adding a “may fail” [EINVAL] error  
37904 condition for the L\_SENDFD command.

37905 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

37906 The *ioctl()* function is marked obsolescent.

37907 **NAME**

37908 isalnum, isalnum\_l — test for an alphanumeric character

37909 **SYNOPSIS**

```
37910 #include <ctype.h>
37911 int isalnum(int c);
37912 CX int isalnum_l(int c, locale_t locale);
```

37913 **DESCRIPTION**

37914 CX For *isalnum()*: The functionality described on this reference page is aligned with the ISO C  
 37915 standard. Any conflict between the requirements described here and the ISO C standard is  
 37916 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

37917 CX The *isalnum()* and *isalnum\_l()* functions shall test whether *c* is a character of class **alpha** or  
 37918 CX **digit** in the current locale of the process, or in the locale represented by *locale*, respectively; see  
 37919 XBD Chapter 7 (on page 135).

37920 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
 37921 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
 37922 behavior is undefined.

37923 **RETURN VALUE**

37924 CX The *isalnum()* and *isalnum\_l()* functions shall return non-zero if *c* is an alphanumeric character;  
 37925 otherwise, they shall return 0.

37926 **ERRORS**

37927 The *isalnum\_l()* function may fail if:

37928 CX [EINVAL] *locale* is not a valid locale object handle.

37929 **EXAMPLES**

37930 None.

37931 **APPLICATION USAGE**

37932 To ensure applications portability, especially across natural languages, only these functions and  
 37933 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 37934 classification.

37935 **RATIONALE**

37936 None.

37937 **FUTURE DIRECTIONS**

37938 None.

37939 **SEE ALSO**

37940 *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 37941 *isxdigit()*, *setlocale()*, *uselocale()*

37942 XBD Chapter 7 (on page 135), <ctype.h>, <stdio.h>

37943 **CHANGE HISTORY**

37944 First released in Issue 1. Derived from Issue 1 of the SVID.

37945 **Issue 6**

37946 The normative text is updated to avoid use of the term “must” for application requirements.

## isalnum()

37947 **Issue 7**

37948

37949

The *isalnum\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

37950 **NAME**

37951 isalpha, isalpha\_l — test for an alphabetic character

37952 **SYNOPSIS**

```
37953 #include <ctype.h>
37954 int isalpha(int c);
37955 CX int isalpha_l(int c, locale_t locale);
```

37956 **DESCRIPTION**

37957 CX For *isalpha()*: The functionality described on this reference page is aligned with the ISO C  
 37958 standard. Any conflict between the requirements described here and the ISO C standard is  
 37959 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

37960 CX The *isalpha()* and *isalpha\_l()* functions shall test whether *c* is a character of class **alpha** in the  
 37961 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD  
 37962 Chapter 7 (on page 135).

37963 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
 37964 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
 37965 behavior is undefined.

37966 **RETURN VALUE**

37967 CX The *isalpha()* and *isalpha\_l()* functions shall return non-zero if *c* is an alphabetic character;  
 37968 otherwise, they shall return 0.

37969 **ERRORS**

37970 The *isalpha\_l()* function may fail if:

37971 CX [EINVAL] *locale* is not a valid locale object handle.

37972 **EXAMPLES**

37973 None.

37974 **APPLICATION USAGE**

37975 To ensure applications portability, especially across natural languages, only these functions and  
 37976 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 37977 classification.

37978 **RATIONALE**

37979 None.

37980 **FUTURE DIRECTIONS**

37981 None.

37982 **SEE ALSO**

37983 *isalnum()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 37984 *isxdigit()*, *setlocale()*, *uselocale()*

37985 XBD Chapter 7 (on page 135), [<ctype.h>](#), [<locale.h>](#), [<stdio.h>](#)

37986 **CHANGE HISTORY**

37987 First released in Issue 1. Derived from Issue 1 of the SVID.

37988 **Issue 6**

37989 The normative text is updated to avoid use of the term “must” for application requirements.

## isalpha()

37990 **Issue 7**

37991

37992

The *isalpha\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

37993 **NAME**

37994 isascii — test for a 7-bit US-ASCII character

37995 **SYNOPSIS**

```
37996 OB XSI #include <ctype.h>  
37997 int isascii(int c);
```

37998 **DESCRIPTION**37999 The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.38000 The *isascii()* function is defined on all integer values.38001 **RETURN VALUE**38002 The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and  
38003 octal 0177 inclusive; otherwise, it shall return 0.38004 **ERRORS**

38005 No errors are defined.

38006 **EXAMPLES**

38007 None.

38008 **APPLICATION USAGE**38009 The *isascii()* function cannot be used portably in a localized application.38010 **RATIONALE**

38011 None.

38012 **FUTURE DIRECTIONS**38013 The *isascii()* function may be removed in a future version.38014 **SEE ALSO**38015 XBD [<ctype.h>](#)38016 **CHANGE HISTORY**

38017 First released in Issue 1. Derived from Issue 1 of the SVID.

38018 **Issue 7**38019 The *isascii()* function is marked obsolescent.

**isastream()**38020 **NAME**

38021 isastream — test a file descriptor (STREAMS)

38022 **SYNOPSIS**

```
38023 OB XSR #include <stropts.h>  
38024 int isastream(int fildes);
```

38025 **DESCRIPTION**

38026 The *isastream()* function shall test whether *fildes*, an open file descriptor, is associated with a  
38027 STREAMS-based file.

38028 **RETURN VALUE**

38029 Upon successful completion, *isastream()* shall return 1 if *fildes* refers to a STREAMS-based file  
38030 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.

38031 **ERRORS**

38032 The *isastream()* function shall fail if:

38033 [EBADF] The *fildes* argument is not a valid open file descriptor.

38034 **EXAMPLES**

38035 None.

38036 **APPLICATION USAGE**

38037 None.

38038 **RATIONALE**

38039 None.

38040 **FUTURE DIRECTIONS**

38041 The *isastream()* function may be removed in a future version.

38042 **SEE ALSO**

38043 XBD <stropts.h>

38044 **CHANGE HISTORY**

38045 First released in Issue 4, Version 2.

38046 **Issue 5**

38047 Moved from X/OPEN UNIX extension to BASE.

38048 **Issue 7**

38049 The *isastream()* function is marked obsolescent.

38050 **NAME**

38051 isatty — test for a terminal device

38052 **SYNOPSIS**

38053 #include &lt;unistd.h&gt;

38054 int isatty(int *fildev*);38055 **DESCRIPTION**38056 The *isatty()* function shall test whether *fildev*, an open file descriptor, is associated with a  
38057 terminal device.38058 **RETURN VALUE**38059 The *isatty()* function shall return 1 if *fildev* is associated with a terminal; otherwise, it shall return  
38060 0 and may set *errno* to indicate the error.38061 **ERRORS**38062 The *isatty()* function may fail if:38063 [EBADF] The *fildev* argument is not a valid open file descriptor.38064 [ENOTTY] The file associated with the *fildev* argument is not a terminal.38065 **EXAMPLES**

38066 None.

38067 **APPLICATION USAGE**38068 The *isatty()* function does not necessarily indicate that a human being is available for interaction  
38069 via *fildev*. It is quite possible that non-terminal devices are connected to the communications  
38070 line.38071 **RATIONALE**

38072 None.

38073 **FUTURE DIRECTIONS**

38074 None.

38075 **SEE ALSO**

38076 XBD &lt;unistd.h&gt;

38077 **CHANGE HISTORY**

38078 First released in Issue 1. Derived from Issue 1 of the SVID.

38079 **Issue 6**38080 The following new requirements on POSIX implementations derive from alignment with the  
38081 Single UNIX Specification:

- 38082 • The optional setting of *errno* to indicate an error is added.
- 38083 • The [EBADF] and [ENOTTY] optional error conditions are added.

38084 **Issue 7**

38085 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

**isblank()**38086 **NAME**38087 `isblank, isblank_l` — test for a blank character38088 **SYNOPSIS**

```
38089 #include <ctype.h>
38090 int isblank(int c);
38091 CX int isblank_l(int c, locale_t locale);
```

38092 **DESCRIPTION**

38093 CX For `isblank()`: The functionality described on this reference page is aligned with the ISO C  
 38094 standard. Any conflict between the requirements described here and the ISO C standard is  
 38095 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38096 CX The `isblank()` and `isblank_l()` functions shall test whether `c` is a character of class **blank** in the  
 38097 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38098 Chapter 7 (on page 135).

38099 The `c` argument is a type **int**, the value of which the application shall ensure is a character  
 38100 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38101 any other value, the behavior is undefined.

38102 **RETURN VALUE**

38103 CX The `isblank()` and `isblank_l()` functions shall return non-zero if `c` is a <blank>; otherwise, they  
 38104 shall return 0.

38105 **ERRORS**

38106 The `isblank_l()` function may fail if:

38107 CX [EINVAL] `locale` is not a valid locale object handle.

38108 **EXAMPLES**

38109 None.

38110 **APPLICATION USAGE**

38111 To ensure applications portability, especially across natural languages, only these functions and  
 38112 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38113 classification.

38114 **RATIONALE**

38115 None.

38116 **FUTURE DIRECTIONS**

38117 None.

38118 **SEE ALSO**

38119 `isalnum()`, `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,  
 38120 `isxdigit()`, `setlocale()`, `uselocale()`

38121 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38122 **CHANGE HISTORY**

38123 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38124 **Issue 7**

38125 The `isblank_l()` function is added from The Open Group Technical Standard, 2006, Extended API  
 38126 Set Part 4.

38127 **NAME**

38128           iscntrl, iscntrl\_l — test for a control character

38129 **SYNOPSIS**

```
38130           #include <ctype.h>
38131           int iscntrl(int c);
38132 CX        int iscntrl_l(int c, locale_t locale);
```

38133 **DESCRIPTION**

38134 CX       For *iscntrl()*: The functionality described on this reference page is aligned with the ISO C  
38135 standard. Any conflict between the requirements described here and the ISO C standard is  
38136 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38137 CX       The *iscntrl()* and *iscntrl\_l()* functions shall test whether *c* is a character of class **cntrl** in the  
38138 CX       current locale of the process, or in the locale represented by *locale*, respectively; see XBD  
38139 Chapter 7 (on page 135).

38140       The *c* argument is a type **int**, the value of which the application shall ensure is a character  
38141 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
38142 any other value, the behavior is undefined.

38143 **RETURN VALUE**

38144 CX       The *iscntrl()* and *iscntrl\_l()* functions shall return non-zero if *c* is a control character; otherwise,  
38145 they shall return 0.

38146 **ERRORS**

38147       The *iscntrl\_l()* function may fail if:

38148 CX       [EINVAL] *locale* is not a valid locale object handle.

38149 **EXAMPLES**

38150       None.

38151 **APPLICATION USAGE**

38152       To ensure applications portability, especially across natural languages, only these functions and  
38153 the functions in the reference pages listed in the SEE ALSO section should be used for character  
38154 classification.

38155 **RATIONALE**

38156       None.

38157 **FUTURE DIRECTIONS**

38158       None.

38159 **SEE ALSO**

38160       *isalnum()*, *isalpha()*, *isblank()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,  
38161 *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

38162       XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38163 **CHANGE HISTORY**

38164       First released in Issue 1. Derived from Issue 1 of the SVID.

38165 **Issue 6**

38166       The normative text is updated to avoid use of the term “must” for application requirements.

## iscntrl()

38167 **Issue 7**

38168 The *iscntrl\_I()* function is added from The Open Group Technical Standard, 2006, Extended API  
38169 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38170 **NAME**

38171 isdigit, isdigit\_l — test for a decimal digit

38172 **SYNOPSIS**

```
38173 #include <ctype.h>
38174 int isdigit(int c);
38175 CX int isdigit_l(int c, locale_t locale);
```

38176 **DESCRIPTION**

38177 CX For *isdigit()*: The functionality described on this reference page is aligned with the ISO C  
 38178 standard. Any conflict between the requirements described here and the ISO C standard is  
 38179 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38180 CX The *isdigit()* and *isdigit\_l()* functions shall test whether *c* is a character of class **digit** in the  
 38181 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD  
 38182 Chapter 7 (on page 135).

38183 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 38184 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38185 any other value, the behavior is undefined.

38186 **RETURN VALUE**

38187 CX The *isdigit()* and *isdigit\_l()* functions shall return non-zero if *c* is a decimal digit; otherwise,  
 38188 they shall return 0.

38189 **ERRORS**

38190 The *isdigit\_l()* function may fail if:

38191 CX [EINVAL] *locale* is not a valid locale object handle.

38192 **EXAMPLES**

38193 None.

38194 **APPLICATION USAGE**

38195 To ensure applications portability, especially across natural languages, only these functions and  
 38196 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38197 classification.

38198 **RATIONALE**

38199 None.

38200 **FUTURE DIRECTIONS**

38201 None.

38202 **SEE ALSO**

38203 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,  
 38204 *isupper()*, *isxdigit()*

38205 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38206 **CHANGE HISTORY**

38207 First released in Issue 1. Derived from Issue 1 of the SVID.

38208 **Issue 6**

38209 The normative text is updated to avoid use of the term “must” for application requirements.

## isdigit()

38210 **Issue 7**

38211 The *isdigit\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38212 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38213 **NAME**

38214 isfinite — test for finite value

38215 **SYNOPSIS**

38216 #include &lt;math.h&gt;

38217 int isfinite(real-floating x);

38218 **DESCRIPTION**

38219 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 38220 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38221 volume of POSIX.1-2008 defers to the ISO C standard.

38222 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or  
 38223 normal, and not infinite or NaN). First, an argument represented in a format wider than its  
 38224 semantic type is converted to its semantic type. Then determination is based on the type of the  
 38225 argument.

38226 **RETURN VALUE**38227 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.38228 **ERRORS**

38229 No errors are defined.

38230 **EXAMPLES**

38231 None.

38232 **APPLICATION USAGE**

38233 None.

38234 **RATIONALE**

38235 None.

38236 **FUTURE DIRECTIONS**

38237 None.

38238 **SEE ALSO**38239 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

38240 XBD &lt;math.h&gt;

38241 **CHANGE HISTORY**

38242 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**isgraph()**38243 **NAME**38244 `isgraph, isgraph_l` — test for a visible character38245 **SYNOPSIS**

```
38246 #include <ctype.h>
38247 int isgraph(int c);
38248 CX int isgraph_l(int c, locale_t locale);
```

38249 **DESCRIPTION**

38250 CX For `isgraph()`: The functionality described on this reference page is aligned with the ISO C  
 38251 standard. Any conflict between the requirements described here and the ISO C standard is  
 38252 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38253 CX The `isgraph()` and `isgraph_l()` functions shall test whether `c` is a character of class **graph** in the  
 38254 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38255 Chapter 7 (on page 135).

38256 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 38257 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38258 any other value, the behavior is undefined.

38259 **RETURN VALUE**

38260 CX The `isgraph()` and `isgraph_l()` functions shall return non-zero if `c` is a character with a visible  
 38261 representation; otherwise, they shall return 0.

38262 **ERRORS**

38263 The `isgraph_l()` function may fail if:

38264 CX [EINVAL] `locale` is not a valid locale object handle.

38265 **EXAMPLES**

38266 None.

38267 **APPLICATION USAGE**

38268 To ensure applications portability, especially across natural languages, only these functions and  
 38269 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38270 classification.

38271 **RATIONALE**

38272 None.

38273 **FUTURE DIRECTIONS**

38274 None.

38275 **SEE ALSO**

38276 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,  
 38277 `isxdigit()`, `setlocale()`, `uselocale()`

38278 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38279 **CHANGE HISTORY**

38280 First released in Issue 1. Derived from Issue 1 of the SVID.

38281 **Issue 6**

38282 The normative text is updated to avoid use of the term “must” for application requirements.

38283 **Issue 7**

38284 The *isgraph\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38285 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**isgreater()**38286 **NAME**38287 `isgreater` — test if  $x$  greater than  $y$ 38288 **SYNOPSIS**38289 `#include <math.h>`38290 `int isgreater(real-floating  $x$ , real-floating  $y$ );`38291 **DESCRIPTION**

38292 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 38293 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38294 volume of POSIX.1-2008 defers to the ISO C standard.

38295 The `isgreater()` macro shall determine whether its first argument is greater than its second  
 38296 argument. The value of `isgreater( $x$ ,  $y$ )` shall be equal to  $(x) > (y)$ ; however, unlike  $(x) > (y)$ ,  
 38297 `isgreater( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered.

38298 **RETURN VALUE**38299 Upon successful completion, the `isgreater()` macro shall return the value of  $(x) > (y)$ .38300 If  $x$  or  $y$  is NaN, 0 shall be returned.38301 **ERRORS**

38302 No errors are defined.

38303 **EXAMPLES**

38304 None.

38305 **APPLICATION USAGE**

38306 The relational and equality operators support the usual mathematical relationships between  
 38307 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
 38308 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
 38309 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
 38310 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
 38311 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
 38312 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
 38313 indicates that the argument shall be an expression of **real-floating** type.

38314 **RATIONALE**

38315 None.

38316 **FUTURE DIRECTIONS**

38317 None.

38318 **SEE ALSO**38319 `isgreaterequal()`, `isless()`, `islessequal()`, `islessgreater()`, `isunordered()`38320 **XBD** `<math.h>`38321 **CHANGE HISTORY**

38322 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38323 **NAME**38324 `isgreaterequal` — test if  $x$  is greater than or equal to  $y$ 38325 **SYNOPSIS**38326 `#include <math.h>`38327 `int isgreaterequal(real-floating  $x$ , real-floating  $y$ );`38328 **DESCRIPTION**38329 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38330 conflict between the requirements described here and the ISO C standard is unintentional. This  
38331 volume of POSIX.1-2008 defers to the ISO C standard.38332 The `isgreaterequal()` macro shall determine whether its first argument is greater than or equal to  
38333 its second argument. The value of `isgreaterequal( $x$ ,  $y$ )` shall be equal to  $(x) \geq (y)$ ; however, unlike  
38334  $(x) \geq (y)$ , `isgreaterequal( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are  
38335 unordered.38336 **RETURN VALUE**38337 Upon successful completion, the `isgreaterequal()` macro shall return the value of  $(x) \geq (y)$ .38338 If  $x$  or  $y$  is NaN, 0 shall be returned.38339 **ERRORS**

38340 No errors are defined.

38341 **EXAMPLES**

38342 None.

38343 **APPLICATION USAGE**38344 The relational and equality operators support the usual mathematical relationships between  
38345 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
38346 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
38347 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
38348 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
38349 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
38350 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
38351 indicates that the argument shall be an expression of **real-floating** type.38352 **RATIONALE**

38353 None.

38354 **FUTURE DIRECTIONS**

38355 None.

38356 **SEE ALSO**38357 `isgreater()`, `isless()`, `islessequal()`, `islessgreater()`, `isunordered()`38358 XBD `<math.h>`38359 **CHANGE HISTORY**

38360 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**isinf()**38361 **NAME**

38362           isinf — test for infinity

38363 **SYNOPSIS**

38364           #include &lt;math.h&gt;

38365           int isinf(real-floating x);

38366 **DESCRIPTION**

38367 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
38368 conflict between the requirements described here and the ISO C standard is unintentional. This  
38369 volume of POSIX.1-2008 defers to the ISO C standard.

38370       The *isinf()* macro shall determine whether its argument value is an infinity (positive or  
38371 negative). First, an argument represented in a format wider than its semantic type is converted  
38372 to its semantic type. Then determination is based on the type of the argument.

38373 **RETURN VALUE**38374       The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.38375 **ERRORS**

38376       No errors are defined.

38377 **EXAMPLES**

38378       None.

38379 **APPLICATION USAGE**

38380       None.

38381 **RATIONALE**

38382       None.

38383 **FUTURE DIRECTIONS**

38384       None.

38385 **SEE ALSO**38386       [fpclassify\(\)](#), [isfinite\(\)](#), [isnan\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)38387       XBD [<math.h>](#)38388 **CHANGE HISTORY**

38389       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38390 **NAME**38391 isless — test if  $x$  is less than  $y$ 38392 **SYNOPSIS**

38393 #include &lt;math.h&gt;

38394 int isless(real-floating  $x$ , real-floating  $y$ );38395 **DESCRIPTION**38396 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38397 conflict between the requirements described here and the ISO C standard is unintentional. This  
38398 volume of POSIX.1-2008 defers to the ISO C standard.38399 The *isless()* macro shall determine whether its first argument is less than its second argument.  
38400 The value of *isless*( $x$ ,  $y$ ) shall be equal to  $(x) < (y)$ ; however, unlike  $(x) < (y)$ , *isless*( $x$ ,  $y$ ) shall not  
38401 raise the invalid floating-point exception when  $x$  and  $y$  are unordered.38402 **RETURN VALUE**38403 Upon successful completion, the *isless()* macro shall return the value of  $(x) < (y)$ .38404 If  $x$  or  $y$  is NaN, 0 shall be returned.38405 **ERRORS**

38406 No errors are defined.

38407 **EXAMPLES**

38408 None.

38409 **APPLICATION USAGE**38410 The relational and equality operators support the usual mathematical relationships between  
38411 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
38412 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
38413 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
38414 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
38415 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
38416 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
38417 indicates that the argument shall be an expression of **real-floating** type.38418 **RATIONALE**

38419 None.

38420 **FUTURE DIRECTIONS**

38421 None.

38422 **SEE ALSO**38423 *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*

38424 XBD &lt;math.h&gt;

38425 **CHANGE HISTORY**

38426 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**islessequal()**38427 **NAME**38428 `islessequal` — test if  $x$  is less than or equal to  $y$ 38429 **SYNOPSIS**38430 `#include <math.h>`38431 `int islessequal(real-floating  $x$ , real-floating  $y$ );`38432 **DESCRIPTION**38433 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38434 conflict between the requirements described here and the ISO C standard is unintentional. This  
38435 volume of POSIX.1-2008 defers to the ISO C standard.38436 The `islessequal()` macro shall determine whether its first argument is less than or equal to its  
38437 second argument. The value of `islessequal( $x$ ,  $y$ )` shall be equal to  $(x) \leq (y)$ ; however, unlike  
38438  $(x) \leq (y)$ , `islessequal( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are  
38439 unordered.38440 **RETURN VALUE**38441 Upon successful completion, the `islessequal()` macro shall return the value of  $(x) \leq (y)$ .38442 If  $x$  or  $y$  is NaN, 0 shall be returned.38443 **ERRORS**

38444 No errors are defined.

38445 **EXAMPLES**

38446 None.

38447 **APPLICATION USAGE**38448 The relational and equality operators support the usual mathematical relationships between  
38449 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
38450 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
38451 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
38452 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
38453 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
38454 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
38455 indicates that the argument shall be an expression of **real-floating** type.38456 **RATIONALE**

38457 None.

38458 **FUTURE DIRECTIONS**

38459 None.

38460 **SEE ALSO**38461 `isgreater()`, `isgreaterequal()`, `isless()`, `islessgreater()`, `isunordered()`38462 XBD `<math.h>`38463 **CHANGE HISTORY**

38464 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38465 **NAME**38466 `islessgreater` — test if  $x$  is less than or greater than  $y$ 38467 **SYNOPSIS**38468 `#include <math.h>`38469 `int islessgreater(real-floating  $x$ , real-floating  $y$ );`38470 **DESCRIPTION**38471 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38472 conflict between the requirements described here and the ISO C standard is unintentional. This  
38473 volume of POSIX.1-2008 defers to the ISO C standard.38474 The `islessgreater()` macro shall determine whether its first argument is less than or greater than  
38475 its second argument. The `islessgreater( $x$ ,  $y$ )` macro is similar to  $(x) < (y) \parallel (x) > (y)$ ; however,  
38476 `islessgreater( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered  
38477 (nor shall it evaluate  $x$  and  $y$  twice).38478 **RETURN VALUE**38479 Upon successful completion, the `islessgreater()` macro shall return the value of  
38480  $(x) < (y) \parallel (x) > (y)$ .38481 If  $x$  or  $y$  is NaN, 0 shall be returned.38482 **ERRORS**

38483 No errors are defined.

38484 **EXAMPLES**

38485 None.

38486 **APPLICATION USAGE**38487 The relational and equality operators support the usual mathematical relationships between  
38488 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
38489 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
38490 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
38491 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
38492 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
38493 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
38494 indicates that the argument shall be an expression of **real-floating** type.38495 **RATIONALE**

38496 None.

38497 **FUTURE DIRECTIONS**

38498 None.

38499 **SEE ALSO**38500 `isgreater()`, `isgreaterequal()`, `isless()`, `islessequal()`, `isunordered()`38501 XBD `<math.h>`38502 **CHANGE HISTORY**

38503 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**islower()**38504 **NAME**38505 `islower, islower_l` — test for a lowercase letter38506 **SYNOPSIS**

```
38507     #include <ctype.h>
38508     int islower(int c);
38509 CX    int islower_l(int c, locale_t locale);
```

38510 **DESCRIPTION**

38511 CX For `islower()`: The functionality described on this reference page is aligned with the ISO C  
 38512 standard. Any conflict between the requirements described here and the ISO C standard is  
 38513 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38514 CX The `islower()` and `islower_l()` functions shall test whether `c` is a character of class **lower** in the  
 38515 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38516 Chapter 7 (on page 135).

38517 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 38518 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38519 any other value, the behavior is undefined.

38520 **RETURN VALUE**

38521 CX The `islower()` and `islower_l()` functions shall return non-zero if `c` is a lowercase letter; otherwise,  
 38522 they shall return 0.

38523 **ERRORS**

38524 The `islower_l()` function may fail if:

38525 CX [EINVAL] `locale` is not a valid locale object handle.

38526 **EXAMPLES**38527 **Testing for a Lowercase Letter**

38528 Two examples follow, the first using `islower()`, the second using multiple concurrent locales and  
 38529 `islower_l()`.

38530 The examples test whether the value is a lowercase letter, based on the locale of the user, then  
 38531 use it as part of a key value.

```
38532     /* Example 1 -- using islower() */
38533     #include <ctype.h>
38534     #include <stdlib.h>
38535     #include <locale.h>
38536     .
38537     char *keystr;
38538     int elementlen, len;
38539     char c;
38540     ...
38541     setlocale(LC_ALL, "");
38542     ...
38543     len = 0;
38544     while (len < elementlen) {
38545         c = (char) (rand() % 256);
38546         ...
38547         if (islower(c))
```

```

38548         keystr[len++] = c;
38549     }
38550     ...
38551     /* Example 2 -- using islower_l() */
38552     #include <ctype.h>
38553     #include <stdlib.h>
38554     #include <locale.h>
38555     ...
38556     char *keystr;
38557     int elementlen, len;
38558     char c;
38559     ...
38560     locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
38561     ...
38562     len = 0;
38563     while (len < elementlen) {
38564         c = (char) (rand() % 256);
38565         ...
38566         if (islower_l(c, loc))
38567             keystr[len++] = c;
38568     }
38569     ...

```

**38570 APPLICATION USAGE**

38571 To ensure applications portability, especially across natural languages, only these functions and  
38572 the functions in the reference pages listed in the SEE ALSO section should be used for character  
38573 classification.

**38574 RATIONALE**

38575 None.

**38576 FUTURE DIRECTIONS**

38577 None.

**38578 SEE ALSO**

38579 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
38580 *isxdigit()*, *setlocale()*, *uselocale()*

38581 XBD Chapter 7 (on page 135), [<ctype.h>](#), [<locale.h>](#)

**38582 CHANGE HISTORY**

38583 First released in Issue 1. Derived from Issue 1 of the SVID.

**38584 Issue 6**

38585 The normative text is updated to avoid use of the term “must” for application requirements.

38586 An example is added.

**38587 Issue 7**

38588 The *islower\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38589 Set Part 4.

**isnan()**38590 **NAME**38591            **isnan** — test for a NaN38592 **SYNOPSIS**

38593            #include &lt;math.h&gt;

38594            int isnan(real-floating x);

38595 **DESCRIPTION**

38596 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 38597 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38598 volume of POSIX.1-2008 defers to the ISO C standard.

38599        The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument  
 38600 represented in a format wider than its semantic type is converted to its semantic type. Then  
 38601 determination is based on the type of the argument.

38602 **RETURN VALUE**38603        The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.38604 **ERRORS**

38605        No errors are defined.

38606 **EXAMPLES**

38607        None.

38608 **APPLICATION USAGE**

38609        None.

38610 **RATIONALE**

38611        None.

38612 **FUTURE DIRECTIONS**

38613        None.

38614 **SEE ALSO**38615        [fpclassify\(\)](#), [isfinite\(\)](#), [isinf\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)38616        XBD [<math.h>](#)38617 **CHANGE HISTORY**

38618        First released in Issue 3.

38619 **Issue 5**

38620        The DESCRIPTION is updated to indicate the return value when NaN is not supported. This  
 38621 text was previously published in the APPLICATION USAGE section.

38622 **Issue 6**

38623        Re-written for alignment with the ISO/IEC 9899:1999 standard.

38624 **NAME**

38625           isnormal — test for a normal value

38626 **SYNOPSIS**

38627           #include &lt;math.h&gt;

38628           int isnormal(real-floating x);

38629 **DESCRIPTION**38630 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
38631 conflict between the requirements described here and the ISO C standard is unintentional. This  
38632 volume of POSIX.1-2008 defers to the ISO C standard.38633       The *isnormal()* macro shall determine whether its argument value is normal (neither zero,  
38634 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its  
38635 semantic type is converted to its semantic type. Then determination is based on the type of the  
38636 argument.38637 **RETURN VALUE**38638       The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal  
38639 value.38640 **ERRORS**

38641       No errors are defined.

38642 **EXAMPLES**

38643       None.

38644 **APPLICATION USAGE**

38645       None.

38646 **RATIONALE**

38647       None.

38648 **FUTURE DIRECTIONS**

38649       None.

38650 **SEE ALSO**38651       *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*

38652       XBD &lt;math.h&gt;

38653 **CHANGE HISTORY**

38654       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**isprint()**38655 **NAME**38656 `isprint, isprint_l` — test for a printable character38657 **SYNOPSIS**

```
38658     #include <ctype.h>
38659     int isprint(int c);
38660 CX    int isprint_l(int c, locale_t locale);
```

38661 **DESCRIPTION**

38662 CX For `isprint()`: The functionality described on this reference page is aligned with the ISO C  
 38663 standard. Any conflict between the requirements described here and the ISO C standard is  
 38664 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38665 CX The `isprint()` and `isprint_l()` functions shall test whether `c` is a character of class **print** in the  
 38666 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38667 Chapter 7 (on page 135).

38668 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 38669 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38670 any other value, the behavior is undefined.

38671 **RETURN VALUE**

38672 CX The `isprint()` and `isprint_l()` functions shall return non-zero if `c` is a printable character;  
 38673 otherwise, they shall return 0.

38674 **ERRORS**

38675 The `isprint_l()` function may fail if:

38676 CX **[EINVAL]** `locale` is not a valid locale object handle.

38677 **EXAMPLES**

38678 None.

38679 **APPLICATION USAGE**

38680 To ensure applications portability, especially across natural languages, only these functions and  
 38681 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38682 classification.

38683 **RATIONALE**

38684 None.

38685 **FUTURE DIRECTIONS**

38686 None.

38687 **SEE ALSO**

38688 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `ispunct()`, `isspace()`, `isupper()`,  
 38689 `isxdigit()`, `setlocale()`, `uselocale()`

38690 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38691 **CHANGE HISTORY**

38692 First released in Issue 1. Derived from Issue 1 of the SVID.

38693 **Issue 6**

38694 The normative text is updated to avoid use of the term “must” for application requirements.

38695 **Issue 7**

38696 The *isprint\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38697 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ispunct()**38698 **NAME**38699 `ispunct, ispunct_l` — test for a punctuation character38700 **SYNOPSIS**

```
38701 #include <ctype.h>
38702 int ispunct(int c);
38703 CX int ispunct_l(int c, locale_t locale);
```

38704 **DESCRIPTION**

38705 CX For `ispunct()`: The functionality described on this reference page is aligned with the ISO C  
 38706 standard. Any conflict between the requirements described here and the ISO C standard is  
 38707 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38708 CX The `ispunct()` and `ispunct_l()` functions shall test whether `c` is a character of class **punct** in the  
 38709 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38710 Chapter 7 (on page 135).

38711 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 38712 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38713 any other value, the behavior is undefined.

38714 **RETURN VALUE**

38715 CX The `ispunct()` and `ispunct_l()` functions shall return non-zero if `c` is a punctuation character;  
 38716 otherwise, they shall return 0.

38717 **ERRORS**

38718 The `ispunct_l()` function may fail if:

38719 CX [EINVAL] `locale` is not a valid locale object handle.

38720 **EXAMPLES**

38721 None.

38722 **APPLICATION USAGE**

38723 To ensure applications portability, especially across natural languages, only these functions and  
 38724 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38725 classification.

38726 **RATIONALE**

38727 None.

38728 **FUTURE DIRECTIONS**

38729 None.

38730 **SEE ALSO**

38731 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `isspace()`, `isupper()`,  
 38732 `isxdigit()`, `setlocale()`, `uselocale()`

38733 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38734 **CHANGE HISTORY**

38735 First released in Issue 1. Derived from Issue 1 of the SVID.

38736 **Issue 6**

38737 The normative text is updated to avoid use of the term “must” for application requirements.

38738 **Issue 7**

38739 The *ispunct\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38740 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**isspace()**38741 **NAME**38742 `isspace, isspace_l` — test for a white-space character38743 **SYNOPSIS**

```
38744 #include <ctype.h>
38745 int isspace(int c);
38746 CX int isspace_l(int c, locale_t locale);
```

38747 **DESCRIPTION**

38748 CX For `isspace()`: The functionality described on this reference page is aligned with the ISO C  
 38749 standard. Any conflict between the requirements described here and the ISO C standard is  
 38750 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38751 CX The `isspace()` and `isspace_l()` functions shall test whether `c` is a character of class **space** in the  
 38752 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 38753 Chapter 7 (on page 135).

38754 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 38755 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38756 any other value, the behavior is undefined.

38757 **RETURN VALUE**

38758 CX The `isspace()` and `isspace_l()` functions shall return non-zero if `c` is a white-space character;  
 38759 otherwise, they shall return 0.

38760 **ERRORS**

38761 The `isspace_l()` function may fail if:

38762 CX [EINVAL] `locale` is not a valid locale object handle.

38763 **EXAMPLES**

38764 None.

38765 **APPLICATION USAGE**

38766 To ensure applications portability, especially across natural languages, only these functions and  
 38767 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38768 classification.

38769 **RATIONALE**

38770 None.

38771 **FUTURE DIRECTIONS**

38772 None.

38773 **SEE ALSO**

38774 `isalnum()`, `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isupper()`,  
 38775 `issdigit()`, `setlocale()`, `uselocale()`

38776 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38777 **CHANGE HISTORY**

38778 First released in Issue 1. Derived from Issue 1 of the SVID.

38779 **Issue 6**

38780 The normative text is updated to avoid use of the term “must” for application requirements.

38781 **Issue 7**

38782 The *isspace\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
38783 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**isunordered()**38784 **NAME**38785 `isunordered` — test if arguments are unordered38786 **SYNOPSIS**38787 `#include <math.h>`38788 `int isunordered(real-floating x, real-floating y);`38789 **DESCRIPTION**38790 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38791 conflict between the requirements described here and the ISO C standard is unintentional. This  
38792 volume of POSIX.1-2008 defers to the ISO C standard.38793 The `isunordered()` macro shall determine whether its arguments are unordered.38794 **RETURN VALUE**38795 Upon successful completion, the `isunordered()` macro shall return 1 if its arguments are  
38796 unordered, and 0 otherwise.38797 If `x` or `y` is NaN, 1 shall be returned.38798 **ERRORS**

38799 No errors are defined.

38800 **EXAMPLES**

38801 None.

38802 **APPLICATION USAGE**38803 The relational and equality operators support the usual mathematical relationships between  
38804 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
38805 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
38806 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
38807 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
38808 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
38809 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
38810 indicates that the argument shall be an expression of **real-floating** type.38811 **RATIONALE**

38812 None.

38813 **FUTURE DIRECTIONS**

38814 None.

38815 **SEE ALSO**38816 `isgreater()`, `isgreaterequal()`, `isless()`, `islessequal()`, `islessgreater()`38817 XBD `<math.h>`38818 **CHANGE HISTORY**

38819 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38820 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN  
38821 VALUE section when `x` or `y` is NaN.

38822 **NAME**

38823 isupper, isupper\_l — test for an uppercase letter

38824 **SYNOPSIS**

```
38825 #include <ctype.h>
38826 int isupper(int c);
38827 CX int isupper_l(int c, locale_t locale);
```

38828 **DESCRIPTION**

38829 CX For *isupper()*: The functionality described on this reference page is aligned with the ISO C  
 38830 standard. Any conflict between the requirements described here and the ISO C standard is  
 38831 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38832 CX The *isupper()* and *isupper\_l()* functions shall test whether *c* is a character of class **upper** in the  
 38833 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD  
 38834 Chapter 7 (on page 135).

38835 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 38836 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 38837 any other value, the behavior is undefined.

38838 **RETURN VALUE**

38839 CX The *isupper()* and *isupper\_l()* functions shall return non-zero if *c* is an uppercase letter;  
 38840 otherwise, they shall return 0.

38841 **ERRORS**

38842 The *isupper\_l()* function may fail if:

38843 CX [EINVAL] *locale* is not a valid locale object handle.

38844 **EXAMPLES**

38845 None.

38846 **APPLICATION USAGE**

38847 To ensure applications portability, especially across natural languages, only these functions and  
 38848 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 38849 classification.

38850 **RATIONALE**

38851 None.

38852 **FUTURE DIRECTIONS**

38853 None.

38854 **SEE ALSO**

38855 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,  
 38856 *isxdigit()*, *setlocale()*, *uselocale()*

38857 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38858 **CHANGE HISTORY**

38859 First released in Issue 1. Derived from Issue 1 of the SVID.

38860 **Issue 6**

38861 The normative text is updated to avoid use of the term “must” for application requirements.

## isupper()

38862 **Issue 7**

38863

38864

The *isupper\_1()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38865 **NAME**

38866 iswalnum, iswalnum\_l — test for an alphanumeric wide-character code

38867 **SYNOPSIS**

38868 #include &lt;wctype.h&gt;

38869 int iswalnum(wint\_t wc);

38870 CX int iswalnum\_l(wint\_t wc, locale\_t locale);

38871 **DESCRIPTION**38872 CX For *iswalnum()*: The functionality described on this reference page is aligned with the ISO C  
38873 standard. Any conflict between the requirements described here and the ISO C standard is  
38874 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.38875 CX The *iswalnum()* and *iswalnum\_l()* functions shall test whether *wc* is a wide-character code  
38876 CX representing a character of class **alpha** or **digit** in the current locale of the process, or in the  
38877 locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).38878 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
38879 code corresponding to a valid character in the current locale, or equal to the value of the macro  
38880 WEOF. If the argument has any other value, the behavior is undefined.38881 **RETURN VALUE**38882 CX The *iswalnum()* and *iswalnum\_l()* functions shall return non-zero if *wc* is an alphanumeric  
38883 wide-character code; otherwise, they shall return 0.38884 **ERRORS**38885 The *iswalnum\_l()* function may fail if:38886 CX [EINVAL] *locale* is not a valid locale object handle.38887 **EXAMPLES**

38888 None.

38889 **APPLICATION USAGE**38890 To ensure applications portability, especially across natural languages, only these functions and  
38891 the functions in the reference pages listed in the SEE ALSO section should be used for character  
38892 classification.38893 **RATIONALE**

38894 None.

38895 **FUTURE DIRECTIONS**

38896 None.

38897 **SEE ALSO**38898 *iswalalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
38899 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38900 XBD Chapter 7 (on page 135), &lt;locale.h&gt;, &lt;stdio.h&gt;, &lt;wctype.h&gt;

38901 **CHANGE HISTORY**

38902 First released as a World-wide Portability Interface in Issue 4.

38903 **Issue 5**38904 The following change has been made in this version for alignment with  
38905 ISO/IEC 9899:1990/Amendment 1:1995 (E):

**iswalnum()**

38906 • The SYNOPSIS has been changed to indicate that this function and associated data types  
38907 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38908 **Issue 6**

38909 The normative text is updated to avoid use of the term “must” for application requirements.

38910 **Issue 7**

38911 The *iswalnum\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
38912 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38913 **NAME**

38914 iswalpha, iswalpha\_l — test for an alphabetic wide-character code

38915 **SYNOPSIS**

38916 #include &lt;wctype.h&gt;

38917 int iswalpha(wint\_t wc);

38918 CX int iswalpha\_l(wint\_t wc, locale\_t locale);

38919 **DESCRIPTION**38920 CX For *iswalpha()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.38923 CX The *iswalpha()* and *iswalpha\_l()* functions shall test whether *wc* is a wide-character code representing a character of class **alpha** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).38926 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.38929 **RETURN VALUE**38930 CX The *iswalpha()* and *iswalpha\_l()* functions shall return non-zero if *wc* is an alphabetic wide-character code; otherwise, they shall return 0.38932 **ERRORS**38933 The *iswalpha\_l()* function may fail if:38934 CX [EINVAL] *locale* is not a valid locale object handle.38935 **EXAMPLES**

38936 None.

38937 **APPLICATION USAGE**

38938 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

38941 **RATIONALE**

38942 None.

38943 **FUTURE DIRECTIONS**

38944 None.

38945 **SEE ALSO**38946 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
38947 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38948 XBD Chapter 7 (on page 135), &lt;locale.h&gt;, &lt;wctype.h&gt;

38949 **CHANGE HISTORY**

38950 First released in Issue 4.

38951 **Issue 5**38952 The following change has been made in this version for alignment with  
38953 ISO/IEC 9899:1990/Amendment 1:1995 (E):

**iswalpha()**

38954 • The SYNOPSIS has been changed to indicate that this function and associated data types  
38955 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38956 **Issue 6**

38957 The normative text is updated to avoid use of the term “must” for application requirements.

38958 **Issue 7**

38959 The `iswalpha_l()` function is added from The Open Group Technical Standard, 2006, Extended  
38960 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38961 **NAME**

38962 iswblank, iswblank\_l — test for a blank wide-character code

38963 **SYNOPSIS**

38964 #include &lt;wctype.h&gt;

38965 int iswblank(wint\_t wc);

38966 CX int iswblank\_l(wint\_t wc, locale\_t locale);

38967 **DESCRIPTION**38968 CX For *iswblank()*: The functionality described on this reference page is aligned with the ISO C  
38969 standard. Any conflict between the requirements described here and the ISO C standard is  
38970 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.38971 CX The *iswblank()* and *iswblank\_l()* functions shall test whether *wc* is a wide-character code  
38972 CX representing a character of class **blank** in the current locale of the process, or in the locale  
38973 represented by *locale*, respectively; see XBD Chapter 7 (on page 135).38974 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
38975 code corresponding to a valid character in the current locale, or equal to the value of the macro  
38976 WEOF. If the argument has any other value, the behavior is undefined.38977 **RETURN VALUE**38978 CX The *iswblank()* and *iswblank\_l()* functions shall return non-zero if *wc* is a blank wide-character  
38979 code; otherwise, they shall return 0.38980 **ERRORS**38981 The *iswblank\_l()* function may fail if:38982 CX [EINVAL] *locale* is not a valid locale object handle.38983 **EXAMPLES**

38984 None.

38985 **APPLICATION USAGE**38986 To ensure applications portability, especially across natural languages, only these functions and  
38987 the functions in the reference pages listed in the SEE ALSO section should be used for character  
38988 classification.38989 **RATIONALE**

38990 None.

38991 **FUTURE DIRECTIONS**

38992 None.

38993 **SEE ALSO**38994 *iswalnum()*, *iswalphabeta()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
38995 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38996 XBD Chapter 7 (on page 135), &lt;locale.h&gt;, &lt;wctype.h&gt;

38997 **CHANGE HISTORY**

38998 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38999 **Issue 7**39000 The *iswblank\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
39001 API Set Part 4.

**iswcntrl()**39002 **NAME**39003 `iswcntrl`, `iswcntrl_l` — test for a control wide-character code39004 **SYNOPSIS**39005 `#include <wctype.h>`39006 `int iswcntrl(wint_t wc);`39007 CX `int iswcntrl_l(wint_t wc, locale_t locale);`39008 **DESCRIPTION**39009 CX For `iswcntrl()`: The functionality described on this reference page is aligned with the ISO C  
39010 standard. Any conflict between the requirements described here and the ISO C standard is  
39011 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39012 CX The `iswcntrl()` and `iswcntrl_l()` functions shall test whether `wc` is a wide-character code  
39013 CX representing a character of class **cntrl** in the current locale of the process, or in the locale  
39014 represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39015 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
39016 code corresponding to a valid character in the current locale, or equal to the value of the macro  
39017 WEOF. If the argument has any other value, the behavior is undefined.39018 **RETURN VALUE**39019 CX The `iswcntrl()` and `iswcntrl_l()` functions shall return non-zero if `wc` is a control wide-character  
39020 code; otherwise, they shall return 0.39021 **ERRORS**39022 The `iswcntrl_l()` function may fail if:39023 CX **[EINVAL]** `locale` is not a valid locale object handle.39024 **EXAMPLES**

39025 None.

39026 **APPLICATION USAGE**39027 To ensure applications portability, especially across natural languages, only these functions and  
39028 the functions in the reference pages listed in the SEE ALSO section should be used for character  
39029 classification.39030 **RATIONALE**

39031 None.

39032 **FUTURE DIRECTIONS**

39033 None.

39034 **SEE ALSO**39035 `iswalnum()`, `iswalpha()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,  
39036 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39037 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39038 **CHANGE HISTORY**

39039 First released in Issue 4.

39040 **Issue 5**39041 The following change has been made in this version for alignment with  
39042 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39043                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
39044                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39045 **Issue 6**  
39046                   The normative text is updated to avoid use of the term “must” for application requirements.
- 39047 **Issue 7**  
39048                   The `iswcntrl_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39049                   API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswctype()**39050 **NAME**39051 `iswctype, iswctype_l` — test character for a specified class39052 **SYNOPSIS**

```
39053     #include <wctype.h>
39054     int iswctype(wint_t wc, wctype_t charclass);
39055 CX    int iswctype_l(wint_t wc, wctype_t charclass,
39056                    locale_t locale);
```

39057 **DESCRIPTION**

39058 CX For `iswctype()`: The functionality described on this reference page is aligned with the ISO C  
 39059 standard. Any conflict between the requirements described here and the ISO C standard is  
 39060 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39061 CX The `iswctype()` and `iswctype_l()` functions shall determine whether the wide-character code `wc`  
 39062 CX has the character class `charclass`, returning true or false. The `iswctype()` and `iswctype_l()`  
 39063 functions are defined on WEOF and wide-character codes corresponding to the valid character  
 39064 CX encodings in the current locale, or in the locale represented by `locale`, respectively. If the `wc`  
 39065 argument is not in the domain of the function, the result is undefined. If the value of `charclass` is  
 39066 invalid (that is, not obtained by a call to `wctype()` or `charclass` is invalidated by a subsequent call  
 39067 to `setlocale()` that has affected category `LC_CTYPE`) the result is unspecified.

39068 **RETURN VALUE**

39069 CX The `iswctype()` and `iswctype_l()` functions shall return non-zero (true) if and only if `wc` has the  
 39070 CX property described by `charclass`. If `charclass` is 0, these functions shall return 0.

39071 **ERRORS**39072 The `iswctype_l()` function may fail if:

39073 CX **[EINVAL]** `locale` is not a valid locale object handle.

39074 **EXAMPLES**39075 **Testing for a Valid Character**

```
39076     #include <wctype.h>
39077     ...
39078     int yes_or_no;
39079     wint_t wc;
39080     wctype_t valid_class;
39081     ...
39082     if ((valid_class=wctype("vowel")) == (wctype_t)0)
39083         /* Invalid character class. */
39084         yes_or_no=iswctype(wc, valid_class);
```

39085 **APPLICATION USAGE**

39086 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",  
 39087 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard  
 39088 character classes. In the table below, the functions in the left column are equivalent to the  
 39089 functions in the right column.

39090	<code>iswalnum(wc)</code>	<code>iswctype(wc, wctype("alnum"))</code>
39091	<code>iswalnum_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alnum"), locale)</code>
39092	<code>iswalpha(wc)</code>	<code>iswctype(wc, wctype("alpha"))</code>
39093	<code>iswalpha_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alpha"), locale)</code>
39094	<code>iswblank(wc)</code>	<code>iswctype(wc, wctype("blank"))</code>

39095	<code>iswblank_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("blank"), locale)</code>
39096	<code>iswcntrl(wc)</code>	<code>iswctype(wc, wctype("cntrl"))</code>
39097	<code>iswcntrl_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("cntrl"), locale)</code>
39098	<code>iswdigit(wc)</code>	<code>iswctype(wc, wctype("digit"))</code>
39099	<code>iswdigit_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("digit"), locale)</code>
39100	<code>iswgraph(wc)</code>	<code>iswctype(wc, wctype("graph"))</code>
39101	<code>iswgraph_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("graph"), locale)</code>
39102	<code>iswlower(wc)</code>	<code>iswctype(wc, wctype("lower"))</code>
39103	<code>iswlower_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("lower"), locale)</code>
39104	<code>iswprint(wc)</code>	<code>iswctype(wc, wctype("print"))</code>
39105	<code>iswprint_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("print"), locale)</code>
39106	<code>iswpunct(wc)</code>	<code>iswctype(wc, wctype("punct"))</code>
39107	<code>iswpunct_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("punct"), locale)</code>
39108	<code>iswspace(wc)</code>	<code>iswctype(wc, wctype("space"))</code>
39109	<code>iswspace_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("space"), locale)</code>
39110	<code>iswupper(wc)</code>	<code>iswctype(wc, wctype("upper"))</code>
39111	<code>iswupper_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("upper"), locale)</code>
39112	<code>iswxdigit(wc)</code>	<code>iswctype(wc, wctype("xdigit"))</code>
39113	<code>iswxdigit_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("xdigit"), locale)</code>

**39114 RATIONALE**

39115 None.

**39116 FUTURE DIRECTIONS**

39117 None.

**39118 SEE ALSO**39119 *iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(),*  
39120 *iswspace(), iswupper(), iswxdigit(), setlocale(), uselocale(), wctype()*39121 XBD `<locale.h>`, `<wctype.h>`**39122 CHANGE HISTORY**

39123 First released as World-wide Portability Interfaces in Issue 4.

**39124 Issue 5**39125 The following change has been made in this version for alignment with  
39126 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39127 • The SYNOPSIS has been changed to indicate that this function and associated data types  
39128 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

**39129 Issue 6**39130 The behavior of  $n=0$  is now described.

39131 An example is added.

39132 A new function, `iswblank()`, is added to the list in the APPLICATION USAGE.**39133 Issue 7**39134 The `iswctype_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39135 API Set Part 4.

**iswdigit()**39136 **NAME**39137 `iswdigit, iswdigit_l` — test for a decimal digit wide-character code39138 **SYNOPSIS**39139 `#include <wctype.h>`39140 `int iswdigit(wint_t wc);`39141 CX `int iswdigit_l(wint_t wc, locale_t locale);`39142 **DESCRIPTION**39143 CX For `iswdigit()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39146 CX The `iswdigit()` and `iswdigit_l()` functions shall test whether `wc` is a wide-character code representing a character of class **digit** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39149 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39152 **RETURN VALUE**39153 CX The `iswdigit()` and `iswdigit_l()` functions shall return non-zero if `wc` is a decimal digit wide-character code; otherwise, they shall return 0.39155 **ERRORS**39156 The `iswdigit_l()` function may fail if:39157 CX `[EINVAL]` `locale` is not a valid locale object handle.39158 **EXAMPLES**

39159 None.

39160 **APPLICATION USAGE**

39161 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39164 **RATIONALE**

39165 None.

39166 **FUTURE DIRECTIONS**

39167 None.

39168 **SEE ALSO**39169 `iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39171 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39172 **CHANGE HISTORY**

39173 First released in Issue 4.

39174 **Issue 5**39175 The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):  
39176

- 39177                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
39178                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39179 **Issue 6**  
39180                   The normative text is updated to avoid use of the term “must” for application requirements.
- 39181 **Issue 7**  
39182                   The *iswdigit\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
39183                   API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswgraph()**39184 **NAME**39185 `iswgraph, iswgraph_l` — test for a visible wide-character code39186 **SYNOPSIS**39187 `#include <wctype.h>`39188 `int iswgraph(wint_t wc);`39189 CX `int iswgraph_l(wint_t wc, locale_t locale);`39190 **DESCRIPTION**39191 CX For `iswgraph()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39194 CX The `iswgraph()` and `iswgraph_l()` functions shall test whether `wc` is a wide-character code representing a character of class **graph** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39197 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39200 **RETURN VALUE**39201 CX The `iswgraph()` and `iswgraph_l()` functions shall return non-zero if `wc` is a wide-character code with a visible representation; otherwise, they shall return 0.39203 **ERRORS**39204 The `iswgraph_l()` function may fail if:39205 CX `[EINVAL]` `locale` is not a valid locale object handle.39206 **EXAMPLES**

39207 None.

39208 **APPLICATION USAGE**

39209 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39212 **RATIONALE**

39213 None.

39214 **FUTURE DIRECTIONS**

39215 None.

39216 **SEE ALSO**39217 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39219 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39220 **CHANGE HISTORY**

39221 First released in Issue 4.

39222 **Issue 5**39223 The following change has been made in this version for alignment with  
39224 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39225                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
39226                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

39227 **Issue 6**

39228                   The normative text is updated to avoid use of the term “must” for application requirements.

39229 **Issue 7**

39230                   The `iswgraph_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39231                   API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswlower()**39232 **NAME**39233 `iswlower, iswlower_l` — test for a lowercase letter wide-character code39234 **SYNOPSIS**39235 `#include <wctype.h>`39236 `int iswlower(wint_t wc);`39237 CX `int iswlower_l(wint_t wc, locale_t locale);`39238 **DESCRIPTION**39239 CX For `iswlower()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39242 CX The `iswlower()` and `iswlower_l()` functions shall test whether `wc` is a wide-character code representing a character of class **lower** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39245 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39248 **RETURN VALUE**39249 CX The `iswlower()` and `iswlower_l()` functions shall return non-zero if `wc` is a lowercase letter wide-character code; otherwise, they shall return 0.39251 **ERRORS**39252 The `iswlower_l()` function may fail if:39253 CX `[EINVAL]` `locale` is not a valid locale object handle.39254 **EXAMPLES**

39255 None.

39256 **APPLICATION USAGE**

39257 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39260 **RATIONALE**

39261 None.

39262 **FUTURE DIRECTIONS**

39263 None.

39264 **SEE ALSO**39265 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()` (on page 2162) 139267 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39268 **CHANGE HISTORY**

39269 First released in Issue 4.

39270 **Issue 5**39271 The following change has been made in this version for alignment with  
39272 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39273
- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39274

39275 **Issue 6**

39276 The normative text is updated to avoid use of the term “must” for application requirements.

39277 **Issue 7**

39278 The `iswlower_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39279 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswprint()**39280 **NAME**39281 `iswprint`, `iswprint_l` — test for a printable wide-character code39282 **SYNOPSIS**39283 `#include <wctype.h>`39284 `int iswprint(wint_t wc);`39285 CX `int iswprint_l(wint_t wc, locale_t locale);`39286 **DESCRIPTION**39287 CX For `iswprint()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39290 CX The `iswprint()` and `iswprint_l()` functions shall test whether `wc` is a wide-character code representing a character of class **print** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39293 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39296 **RETURN VALUE**39297 CX The `iswprint()` and `iswprint_l()` functions shall return non-zero if `wc` is a printable wide-character code; otherwise, they shall return 0.39299 **ERRORS**39300 The `iswprint_l()` function may fail if:39301 CX **[EINVAL]** `locale` is not a valid locale object handle.39302 **EXAMPLES**

39303 None.

39304 **APPLICATION USAGE**

39305 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39308 **RATIONALE**

39309 None.

39310 **FUTURE DIRECTIONS**

39311 None.

39312 **SEE ALSO**39313 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswpunct()`,  
39314 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39315 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39316 **CHANGE HISTORY**

39317 First released in Issue 4.

39318 **Issue 5**39319 The following change has been made in this version for alignment with  
39320 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39321 • The SYNOPSIS has been changed to indicate that this function and associated data types  
39322 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

**39323 Issue 6**

39324 The normative text is updated to avoid use of the term “must” for application requirements.

**39325 Issue 7**

39326 The `iswprint_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39327 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswpunct()**39328 **NAME**39329 `iswpunct, iswpunct_l` — test for a punctuation wide-character code39330 **SYNOPSIS**39331 `#include <wctype.h>`39332 `int iswpunct(wint_t wc);`39333 CX `int iswpunct_l(wint_t wc, locale_t locale);`39334 **DESCRIPTION**39335 CX For `iswpunct()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39338 CX The `iswpunct()` and `iswpunct_l()` functions shall test whether `wc` is a wide-character code representing a character of class **punct** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39341 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39344 **RETURN VALUE**39345 CX The `iswpunct()` and `iswpunct_l()` functions shall return non-zero if `wc` is a punctuation wide-character code; otherwise, they shall return 0.39347 **ERRORS**39348 The `iswpunct_l()` function may fail if:39349 CX `[EINVAL]` `locale` is not a valid locale object handle.39350 **EXAMPLES**

39351 None.

39352 **APPLICATION USAGE**

39353 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39356 **RATIONALE**

39357 None.

39358 **FUTURE DIRECTIONS**

39359 None.

39360 **SEE ALSO**39361 `iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39363 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39364 **CHANGE HISTORY**

39365 First released in Issue 4.

39366 **Issue 5**39367 The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):  
39368

- 39369                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
39370                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39371 **Issue 6**  
39372                   The normative text is updated to avoid use of the term “must” for application requirements.
- 39373 **Issue 7**  
39374                   The `iswpunct_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39375                   API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswspace()**39376 **NAME**39377 `iswspace, iswspace_l` — test for a white-space wide-character code39378 **SYNOPSIS**39379 `#include <wctype.h>`39380 `int iswspace(wint_t wc);`39381 CX `int iswspace_l(wint_t wc, locale_t locale);`39382 **DESCRIPTION**39383 CX For `iswspace()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39386 CX The `iswspace()` and `iswspace_l()` functions shall test whether `wc` is a wide-character code representing a character of class **space** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39389 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39392 **RETURN VALUE**39393 CX The `iswspace()` and `iswspace_l()` functions shall return non-zero if `wc` is a white-space wide-character code; otherwise, they shall return 0.39395 **ERRORS**39396 The `iswspace_l()` function may fail if:39397 CX [EINVAL] `locale` is not a valid locale object handle.39398 **EXAMPLES**

39399 None.

39400 **APPLICATION USAGE**

39401 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39404 **RATIONALE**

39405 None.

39406 **FUTURE DIRECTIONS**

39407 None.

39408 **SEE ALSO**39409 `iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39411 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39412 **CHANGE HISTORY**

39413 First released in Issue 4.

39414 **Issue 5**39415 The following change has been made in this version for alignment with  
39416 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39417                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
39418                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39419 **Issue 6**  
39420                   The normative text is updated to avoid use of the term “must” for application requirements.
- 39421 **Issue 7**  
39422                   The `iswspace_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39423                   API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswupper()**39424 **NAME**39425 `iswupper, iswupper_l` — test for an uppercase letter wide-character code39426 **SYNOPSIS**39427 `#include <wctype.h>`39428 `int iswupper(wint_t wc);`39429 CX `int iswupper_l(wint_t wc, locale_t locale);`39430 **DESCRIPTION**39431 CX For `iswupper()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39434 CX The `iswupper()` and `iswupper_l()` functions shall test whether `wc` is a wide-character code representing a character of class **upper** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39437 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39440 **RETURN VALUE**39441 CX The `iswupper()` and `iswupper_l()` functions shall return non-zero if `wc` is an uppercase letter wide-character code; otherwise, they shall return 0.39443 **ERRORS**39444 The `iswupper_l()` function may fail if:39445 CX [EINVAL] `locale` is not a valid locale object handle.39446 **EXAMPLES**

39447 None.

39448 **APPLICATION USAGE**

39449 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39452 **RATIONALE**

39453 None.

39454 **FUTURE DIRECTIONS**

39455 None.

39456 **SEE ALSO**39457 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswxdigit()`, `setlocale()`, `uselocale()`39459 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39460 **CHANGE HISTORY**

39461 First released in Issue 4.

39462 **Issue 5**39463 The following change has been made in this version for alignment with  
39464 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39465
- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39466

39467 **Issue 6**

39468 The normative text is updated to avoid use of the term “must” for application requirements.

39469 **Issue 7**

39470 The `iswupper_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39471 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iswxdigit()**39472 **NAME**39473 `iswxdigit`, `iswxdigit_l` — test for a hexadecimal digit wide-character code39474 **SYNOPSIS**39475 `#include <wctype.h>`39476 `int iswxdigit(wint_t wc);`39477 CX `int iswxdigit_l(wint_t wc, locale_t locale);`39478 **DESCRIPTION**39479 CX For `iswxdigit()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39482 CX The `iswxdigit()` and `iswxdigit_l()` functions shall test whether `wc` is a wide-character code representing a character of class **xdigit** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39485 The `wc` argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39488 **RETURN VALUE**39489 CX The `iswxdigit()` and `iswxdigit_l()` functions shall return non-zero if `wc` is a hexadecimal digit wide-character code; otherwise, they shall return 0.39491 **ERRORS**39492 The `iswxdigit_l()` function may fail if:39493 CX **[EINVAL]** `locale` is not a valid locale object handle.39494 **EXAMPLES**

39495 None.

39496 **APPLICATION USAGE**

39497 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39500 **RATIONALE**

39501 None.

39502 **FUTURE DIRECTIONS**

39503 None.

39504 **SEE ALSO**39505 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`,  
39506 `iswpunct()`, `iswspace()`, `iswupper()`, `setlocale()`, `uselocale()`39507 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39508 **CHANGE HISTORY**

39509 First released in Issue 4.

39510 **Issue 5**39511 The following change has been made in this version for alignment with  
39512 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39513
- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39514

39515 **Issue 6**

39516 The normative text is updated to avoid use of the term “must” for application requirements.

39517 **Issue 7**

39518 The `iswxdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended  
39519 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**isxdigit()**39520 **NAME**39521 `isxdigit, isxdigit_l` — test for a hexadecimal digit39522 **SYNOPSIS**

```
39523     #include <ctype.h>
39524     int isxdigit(int c);
39525 CX    int isxdigit_l(int c, locale_t locale);
```

39526 **DESCRIPTION**

39527 CX For `isxdigit()`: The functionality described on this reference page is aligned with the ISO C  
 39528 standard. Any conflict between the requirements described here and the ISO C standard is  
 39529 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39530 CX The `isxdigit()` and `isxdigit_l()` functions shall test whether `c` is a character of class **xdigit** in the  
 39531 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD  
 39532 Chapter 7 (on page 135).

39533 The `c` argument is an **int**, the value of which the application shall ensure is a character  
 39534 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 39535 any other value, the behavior is undefined.

39536 **RETURN VALUE**

39537 CX The `isxdigit()` and `isxdigit_l()` functions shall return non-zero if `c` is a hexadecimal digit;  
 39538 otherwise, they shall return 0.

39539 **ERRORS**

39540 The `isxdigit_l()` function may fail if:

39541 CX **[EINVAL]** `locale` is not a valid locale object handle.

39542 **EXAMPLES**

39543 None.

39544 **APPLICATION USAGE**

39545 To ensure applications portability, especially across natural languages, only these functions and  
 39546 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 39547 classification.

39548 **RATIONALE**

39549 None.

39550 **FUTURE DIRECTIONS**

39551 None.

39552 **SEE ALSO**

39553 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`,  
 39554 `isupper()`

39555 XBD Chapter 7 (on page 135), `<ctype.h>`

39556 **CHANGE HISTORY**

39557 First released in Issue 1. Derived from Issue 1 of the SVID.

39558 **Issue 6**

39559 The normative text is updated to avoid use of the term “must” for application requirements.

39560 **Issue 7**

39561 The *isxdigit\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
39562 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**j0()**39563 **NAME**39564 `j0, j1, jn` — Bessel functions of the first kind39565 **SYNOPSIS**

```
39566 xSI #include <math.h>
39567 double j0(double x);
39568 double j1(double x);
39569 double jn(int n, double x);
```

39570 **DESCRIPTION**

39571 The `j0()`, `j1()`, and `jn()` functions shall compute Bessel functions of  $x$  of the first kind of orders 0,  
 39572 1, and  $n$ , respectively.

39573 An application wishing to check for error situations should set `errno` to zero and call  
 39574 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 39575 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 39576 zero, an error has occurred.

39577 **RETURN VALUE**

39578 Upon successful completion, these functions shall return the relevant Bessel value of  $x$  of the  
 39579 first kind.

39580 If the  $x$  argument is too large in magnitude, or the correct result would cause underflow, 0 shall  
 39581 be returned and a range error may occur.

39582 If  $x$  is NaN, a NaN shall be returned.

39583 **ERRORS**

39584 These functions may fail if:

39585 **Range Error** The value of  $x$  was too large in magnitude, or an underflow occurred.

39586 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 39587 then `errno` shall be set to [ERANGE]. If the integer expression  
 39588 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow  
 39589 floating-point exception shall be raised.

39590 No other errors shall occur.

39591 **EXAMPLES**

39592 None.

39593 **APPLICATION USAGE**

39594 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &  
 39595 MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

39596 **RATIONALE**

39597 None.

39598 **FUTURE DIRECTIONS**

39599 None.

39600 **SEE ALSO**

39601 `feclearexcept()`, `fetestexcept()`, `isnan()`, `y0()`

39602 XBD Section 4.19 (on page 116), `<math.h>`

39603 **CHANGE HISTORY**

39604 First released in Issue 1. Derived from Issue 1 of the SVID.

39605 **Issue 5**

39606 The DESCRIPTION is updated to indicate how an application should check for an error. This  
39607 text was previously published in the APPLICATION USAGE section.

39608 **Issue 6**

39609 The may fail [EDOM] error is removed for the case for NaN.

39610 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling  
39611 with the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**jrand48()**

System Interfaces

39612 **NAME**

39613           jrand48 — generate a uniformly distributed pseudo-random long signed integer

39614 **SYNOPSIS**

```
39615 XSI       #include <stdlib.h>  
39616           long jrand48(unsigned short xsubi[3]);
```

39617 **DESCRIPTION**39618           Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## System Interfaces

**kill()**39619 **NAME**

39620 kill — send a signal to a process or a group of processes

39621 **SYNOPSIS**

```
39622 CX #include <signal.h>
39623 int kill(pid_t pid, int sig);
```

39624 **DESCRIPTION**

39625 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The  
 39626 signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is  
 39627 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can  
 39628 be used to check the validity of *pid*.

39629 For a process to have permission to send a signal to a process designated by *pid*, unless the  
 39630 sending process has appropriate privileges, the real or effective user ID of the sending process  
 39631 shall match the real or saved set-user-ID of the receiving process.

39632 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

39633 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)  
 39634 whose process group ID is equal to the process group ID of the sender, and for which the process  
 39635 has permission to send a signal.

39636 If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for  
 39637 which the process has permission to send that signal.

39638 If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of  
 39639 system processes) whose process group ID is equal to the absolute value of *pid*, and for which  
 39640 the process has permission to send a signal.

39641 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for  
 39642 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function  
 39643 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending  
 39644 thread before *kill()* returns.

39645 The user ID tests described above shall not be applied when sending SIGCONT to a process that  
 39646 is a member of the same session as the sending process.

39647 An implementation that provides extended security controls may impose further  
 39648 implementation-defined restrictions on the sending of signals, including the null signal. In  
 39649 particular, the system may deny the existence of some or all of the processes specified by *pid*.

39650 The *kill()* function is successful if the process has permission to send *sig* to any of the processes  
 39651 specified by *pid*. If *kill()* fails, no signal shall be sent.

39652 **RETURN VALUE**

39653 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 39654 indicate the error.

39655 **ERRORS**

39656 The *kill()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 39657 | [EINVAL] | The value of the <i>sig</i> argument is an invalid or unsupported signal number.  |
| 39658 | [EPERM]  | The process does not have permission to send the signal to any receiving process. |
| 39659 |          |   |

**kill()**

39660 [ESRCH] No process or process group can be found corresponding to that specified by  
 39661 *pid*.

39662 **EXAMPLES**

39663 None.

39664 **APPLICATION USAGE**

39665 None.

39666 **RATIONALE**

39667 The semantics for permission checking for *kill()* differed between System V and most other  
 39668 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of  
 39669 POSIX.1-2008 agree with System V. Specifically, a set-user-ID process cannot protect itself  
 39670 against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice  
 39671 allows the user who starts an application to send it signals even if it changes its effective user ID.  
 39672 The other semantics give more power to an application that wants to protect itself from the user  
 39673 who ran it.

39674 Some implementations provide semantic extensions to the *kill()* function when the absolute  
 39675 value of *pid* is greater than some maximum, or otherwise special value. Negative values are a  
 39676 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not  
 39677 included in this volume of POSIX.1-2008, although a conforming implementation could provide  
 39678 such an extension.

39679 The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

39680 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the  
 39681 calling process and that signal is not blocked, that signal would be delivered before *kill()*  
 39682 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.  
 39683 However, historical implementations that provide only the *signal()* function make only the  
 39684 weaker guarantee in this volume of POSIX.1-2008, because they only deliver one signal each  
 39685 time a process enters the kernel. Modifications to such implementations to support the  
 39686 *sigaction()* function generally require entry to the kernel following return from a signal-catching  
 39687 function, in order to restore the signal mask. Such modifications have the effect of satisfying the  
 39688 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.  
 39689 The standard developers considered making the stronger requirement except when *signal()* is  
 39690 used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the  
 39691 stronger requirement whenever possible. In practice, the weaker requirement is the same, except  
 39692 in the rare case when two signals arrive during a very short window. This reasoning also applies  
 39693 to a similar requirement for *sigprocmask()*.

39694 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID  
 39695 security checks. This allows a job control shell to continue a job even if processes in the job have  
 39696 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of  
 39697 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any  
 39698 process in the same session regardless of user ID security checks. This is less restrictive than BSD  
 39699 in the sense that ancestor processes (in the same session) can now be the recipient. It is more  
 39700 restrictive than BSD in the sense that descendant processes that form new sessions are now  
 39701 subject to the user ID checks. A similar relaxation of security is not necessary for the other job  
 39702 control signals since those signals are typically sent by the terminal driver in recognition of  
 39703 special characters being typed; the terminal driver bypasses all security checks.

39704 In secure implementations, a process may be restricted from sending a signal to a process having  
 39705 a different security label. In order to prevent the existence or nonexistence of a process from  
 39706 being used as a covert channel, such processes should appear nonexistent to the sender; that is,  
 39707 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

39708 Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a  
 39709 terminated process that has not been waited for by its parent). Some indicate success on such a  
 39710 call (subject to permission checking), while others give an error of [ESRCH]. Since the definition  
 39711 of process lifetime in this volume of POSIX.1-2008 covers inactive processes, the [ESRCH] error  
 39712 as described is inappropriate in this case. In particular, this means that an application cannot  
 39713 have a parent process check for termination of a particular child with *kill()*. (Usually this is done  
 39714 with the null signal; this can be done reliably with *waitpid()*.)

39715 There is some belief that the name *kill()* is misleading, since the function is not always intended  
 39716 to cause process termination. However, the name is common to all historical implementations,  
 39717 and any change would be in conflict with the goal of minimal changes to existing application  
 39718 code.

#### 39719 FUTURE DIRECTIONS

39720 None.

#### 39721 SEE ALSO

39722 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, *wait()*

39723 XBD <signal.h>, <sys/types.h>

#### 39724 CHANGE HISTORY

39725 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 39726 Issue 5

39727 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 39728 Issue 6

39729 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

39730 The following new requirements on POSIX implementations derive from alignment with the  
 39731 Single UNIX Specification:

- 39732 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-  
 39733 user-ID of the calling process is checked in place of its effective user ID. This is a FIPS  
 39734 requirement.
- 39735 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 39736 required for conforming implementations of previous POSIX specifications, it was not  
 39737 required for UNIX applications.
- 39738 • The behavior when *pid* is  $-1$  is now specified. It was previously explicitly unspecified in  
 39739 the POSIX.1-1988 standard.

39740 The normative text is updated to avoid use of the term “must” for application requirements.

39741 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE  
 39742 section.

**killpg()**39743 **NAME**

39744 killpg — send a signal to a process group

39745 **SYNOPSIS**

```
39746 XSI #include <signal.h>
39747 int killpg(pid_t pgrp, int sig);
```

39748 **DESCRIPTION**39749 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.39750 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or  
39751 equal to 1, the behavior of *killpg()* is undefined.39752 **RETURN VALUE**39753 Refer to *kill()*.39754 **ERRORS**39755 Refer to *kill()*.39756 **EXAMPLES**39757 **Sending a Signal to All Other Members of a Process Group**39758 The following example shows how the calling process could send a signal to all other members  
39759 of its process group. To prevent itself from receiving the signal it first makes itself immune to the  
39760 signal by ignoring it.

```
39761 #include <signal.h>
39762 #include <unistd.h>
39763 ...
39764     if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
39765         /* Handle error */;
39766     if (killpg(getpgrp(), SIGUSR1) == -1)
39767         /* Handle error */;
```

39768 **APPLICATION USAGE**

39769 None.

39770 **RATIONALE**

39771 None.

39772 **FUTURE DIRECTIONS**

39773 None.

39774 **SEE ALSO**39775 *getpgid()*, *getpid()*, *kill()*, *raise()*39776 XBD [<signal.h>](#)39777 **CHANGE HISTORY**

39778 First released in Issue 4, Version 2.

39779 **Issue 5**

39780 Moved from X/OPEN UNIX extension to BASE.

39781 **Issue 6**

39782 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the  
39783 EXAMPLES section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**l64a()**39784 **NAME**

39785           l64a — convert a 32-bit integer to a radix-64 ASCII string

39786 **SYNOPSIS**

```
39787 xSI       #include <stdlib.h>  
39788           char *l64a(long value);
```

39789 **DESCRIPTION**39790           Refer to [a64l\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

39791 **NAME**

39792 labs, llabs — return a long integer absolute value

39793 **SYNOPSIS**

39794 #include &lt;stdlib.h&gt;

39795 long labs(long i);

39796 long long llabs(long long i);

39797 **DESCRIPTION**39798 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
39799 conflict between the requirements described here and the ISO C standard is unintentional. This  
39800 volume of POSIX.1-2008 defers to the ISO C standard.39801 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*  
39802 function shall compute the absolute value of the **long long** integer operand *i*. If the result  
39803 cannot be represented, the behavior is undefined.39804 **RETURN VALUE**39805 The *labs()* function shall return the absolute value of the **long** integer operand.39806 The *llabs()* function shall return the absolute value of the **long long** integer operand.39807 **ERRORS**

39808 No errors are defined.

39809 **EXAMPLES**

39810 None.

39811 **APPLICATION USAGE**

39812 None.

39813 **RATIONALE**

39814 None.

39815 **FUTURE DIRECTIONS**

39816 None.

39817 **SEE ALSO**39818 [abs\(\)](#)39819 XBD [<stdlib.h>](#)39820 **CHANGE HISTORY**

39821 First released in Issue 4. Derived from the ISO C standard.

39822 **Issue 6**39823 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.39824 **Issue 7**

39825 SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section.

**lchown()**39826 **NAME**39827 `lchown` — change the owner and group of a symbolic link39828 **SYNOPSIS**39829 `#include <unistd.h>`39830 `int lchown(const char *path, uid_t owner, gid_t group);`39831 **DESCRIPTION**

39832 The `lchown()` function shall be equivalent to `chown()`, except in the case where the named file is a  
 39833 symbolic link. In this case, `lchown()` shall change the ownership of the symbolic link file itself,  
 39834 while `chown()` changes the ownership of the file or directory to which the symbolic link refers.

39835 **RETURN VALUE**

39836 Upon successful completion, `lchown()` shall return 0. Otherwise, it shall return `-1` and set `errno` to  
 39837 indicate an error.

39838 **ERRORS**39839 The `lchown()` function shall fail if:39840 [EACCES] Search permission is denied on a component of the path prefix of `path`.

39841 [EINVAL] The owner or group ID is not a value supported by the implementation.

39842 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 39843 argument.

39844 [ENAMETOOLONG]

39845 The length of a component of a pathname is longer than `{NAME_MAX}`.39846 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

39847 [ENOTDIR] A component of the path prefix is not a directory, or the `path` argument  
 39848 contains at least one non-`<slash>` character and ends with one or more trailing  
 39849 `<slash>` characters and the last pathname component names an existing file  
 39850 that is neither a directory nor a symbolic link to a directory.

39851 [EPERM] The effective user ID does not match the owner of the file and the process does  
 39852 not have appropriate privileges.

39853 [EROFS] The file resides on a read-only file system.

39854 The `lchown()` function may fail if:

39855 [EIO] An I/O error occurred while reading or writing to the file system.

39856 [EINTR] A signal was caught during execution of the function.

39857 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during  
 39858 resolution of the `path` argument.

39859 [ENAMETOOLONG]

39860 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a  
 39861 symbolic link produced an intermediate result with a length that exceeds  
 39862 `{PATH_MAX}`.

39863 **EXAMPLES**39864 **Changing the Current Owner of a File**

39865 The following example shows how to change the ownership of the symbolic link named  
39866 **/modules/pass1** to the user ID associated with "jones" and the group ID associated with "cnd".

39867 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric  
39868 value for the group ID is obtained by using the *getgrnam()* function.

```
39869 #include <sys/types.h>
39870 #include <unistd.h>
39871 #include <pwd.h>
39872 #include <grp.h>

39873 struct passwd *pwd;
39874 struct group *grp;
39875 char *path = "/modules/pass1";
39876 ...
39877 pwd = getpwnam("jones");
39878 grp = getgrnam("cnd");
39879 lchown(path, pwd->pw_uid, grp->gr_gid);
```

39880 **APPLICATION USAGE**

39881 On implementations which support symbolic links as directory entries rather than files, *lchown()*  
39882 may fail.

39883 **RATIONALE**

39884 None.

39885 **FUTURE DIRECTIONS**

39886 None.

39887 **SEE ALSO**

39888 *chown()*, *symlink()*

39889 XBD **<unistd.h>**

39890 **CHANGE HISTORY**

39891 First released in Issue 4, Version 2.

39892 **Issue 5**

39893 Moved from X/OPEN UNIX extension to BASE.

39894 **Issue 6**

39895 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
39896 [ELOOP] error condition is added.

39897 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the  
39898 APPLICATION USAGE.

39899 **Issue 7**

39900 Austin Group Interpretation 1003.1-2001 #143 is applied.

39901 The *lchown()* function is moved from the XSI option to the Base.

39902 The [EOPNOTSUPP] error is removed.

## lchown()

39903  
39904

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

39905 **NAME**

39906 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

39907 **SYNOPSIS**

```
39908 XSI #include <stdlib.h>  
39909 void lcong48(unsigned short param[7]);
```

39910 **DESCRIPTION**39911 Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ldexp()**39912 **NAME**

39913 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

39914 **SYNOPSIS**

```
39915 #include <math.h>
39916 double ldexp(double x, int exp);
39917 float ldexpf(float x, int exp);
39918 long double ldexpl(long double x, int exp);
```

39919 **DESCRIPTION**

39920 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 39921 conflict between the requirements described here and the ISO C standard is unintentional. This  
 39922 volume of POSIX.1-2008 defers to the ISO C standard.

39923 These functions shall compute the quantity  $x * 2^{exp}$ .

39924 An application wishing to check for error situations should set *errno* to zero and call  
 39925 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 39926 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 39927 zero, an error has occurred.

39928 **RETURN VALUE**39929 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power  
39930 *exp*.

39931 If these functions would cause overflow, a range error shall occur and *ldexp()*, *ldexpf()*, and  
 39932 *ldexpl()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (according to the sign of  
 39933 *x*), respectively.

39934 If the correct value would cause underflow, and is not representable, a range error may occur,  
 39935 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

39936 **MX** If *x* is NaN, a NaN shall be returned.39937 If *x* is  $\pm 0$  or  $\pm$ Inf, *x* shall be returned.39938 If *exp* is 0, *x* shall be returned.

39939 If the correct value would cause underflow, and is representable, a range error may occur and  
 39940 the correct value shall be returned.

39941 **ERRORS**

39942 These functions shall fail if:

39943 **Range Error** The result overflows.

39944 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 39945 then *errno* shall be set to [ERANGE]. If the integer expression  
 39946 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 39947 floating-point exception shall be raised.

39948 These functions may fail if:

39949 **Range Error** The result underflows.

39950 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 39951 then *errno* shall be set to [ERANGE]. If the integer expression  
 39952 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 39953 floating-point exception shall be raised.

39954 **EXAMPLES**

39955 None.

39956 **APPLICATION USAGE**39957 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
39958 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.39959 **RATIONALE**

39960 None.

39961 **FUTURE DIRECTIONS**

39962 None.

39963 **SEE ALSO**39964 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [frexp\(\)](#), [isnan\(\)](#)39965 XBD Section 4.19 (on page 116), [<math.h>](#)39966 **CHANGE HISTORY**

39967 First released in Issue 1. Derived from Issue 1 of the SVID.

39968 **Issue 5**39969 The DESCRIPTION is updated to indicate how an application should check for an error. This  
39970 text was previously published in the APPLICATION USAGE section.39971 **Issue 6**39972 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
39973 standard.39974 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
39975 revised to align with the ISO/IEC 9899:1999 standard.39976 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
39977 marked.

**ldiv()**39978 **NAME**39979 `ldiv, lldiv` — compute quotient and remainder of a long division39980 **SYNOPSIS**

```
39981 #include <stdlib.h>
39982
39982 ldiv_t ldiv(long numer, long denom);
39983 lldiv_t lldiv(long long numer, long long denom);
```

39984 **DESCRIPTION**

39985 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 39986 conflict between the requirements described here and the ISO C standard is unintentional. This  
 39987 volume of POSIX.1-2008 defers to the ISO C standard.

39988 These functions shall compute the quotient and remainder of the division of the numerator  
 39989 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**  
 39990 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude  
 39991 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is  
 39992 undefined; otherwise, *quot \* denom + rem* shall equal *numer*.

39993 **RETURN VALUE**

39994 The *ldiv()* function shall return a structure of type **ldiv\_t**, comprising both the quotient and the  
 39995 remainder. The structure shall include the following members, in any order:

```
39996 long quot; /* Quotient */
39997 long rem; /* Remainder */
```

39998 The *lldiv()* function shall return a structure of type **lldiv\_t**, comprising both the quotient and the  
 39999 remainder. The structure shall include the following members, in any order:

```
40000 long long quot; /* Quotient */
40001 long long rem; /* Remainder */
```

40002 **ERRORS**

40003 No errors are defined.

40004 **EXAMPLES**

40005 None.

40006 **APPLICATION USAGE**

40007 None.

40008 **RATIONALE**

40009 None.

40010 **FUTURE DIRECTIONS**

40011 None.

40012 **SEE ALSO**40013 [div\(\)](#)40014 XBD [<stdlib.h>](#)40015 **CHANGE HISTORY**

40016 First released in Issue 4. Derived from the ISO C standard.

40017 **Issue 6**40018 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

40019 **NAME**

40020 lfind — find entry in a linear search table

40021 **SYNOPSIS**

```
40022 XSI #include <search.h>
40023 void *lfind(const void *key, const void *base, size_t *nel,
40024 size_t width, int (*compar)(const void *, const void *));
```

40025 **DESCRIPTION**40026 Refer to *lsearch()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**lgamma()**40027 **NAME**

40028 lgamma, lgammaf, lgammal, signgam — log gamma function

40029 **SYNOPSIS**

```
40030 #include <math.h>
40031 double lgamma(double x);
40032 float lgammaf(float x);
40033 long double lgammal(long double x);
40034 XSI extern int signgam;
```

40035 **DESCRIPTION**

40036 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40037 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40038 volume of POSIX.1-2008 defers to the ISO C standard.

40039 These functions shall compute  $\log_e |\Gamma(x)|$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ . The argument  $x$   
 40040 need not be a non-positive integer ( $\Gamma(x)$  is defined over the reals, except the non-positive  
 40041 integers).

40042 XSI If  $x$  is NaN,  $-\text{Inf}$ , or a negative integer, the value of *signgam* is unspecified.

40043 CX These functions need not be thread-safe.

40044 An application wishing to check for error situations should set *errno* to zero and call  
 40045 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 40046 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 40047 zero, an error has occurred.

40048 **RETURN VALUE**

40049 Upon successful completion, these functions shall return the logarithmic gamma of  $x$ .

40050 If  $x$  is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*  
 40051 shall return  $+\text{HUGE\_VAL}$ ,  $+\text{HUGE\_VALF}$ , and  $+\text{HUGE\_VALL}$ , respectively.

40052 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,  
 40053 and *lgammal()* shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$  (having the same  
 40054 sign as the correct value), respectively.

40055 MX If  $x$  is NaN, a NaN shall be returned.

40056 If  $x$  is 1 or 2,  $+0$  shall be returned.

40057 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

40058 **ERRORS**

40059 These functions shall fail if:

40060 Pole Error The  $x$  argument is a negative integer or zero.

40061 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40062 then *errno* shall be set to [ERANGE]. If the integer expression  
 40063 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 40064 floating-point exception shall be raised.

40065 Range Error The result overflows.

40066 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40067 then *errno* shall be set to [ERANGE]. If the integer expression

40068  $(math\_errhandling$  & MATH\_ERREXCEPT) is non-zero, then the overflow  
40069 floating-point exception shall be raised.

40070 **EXAMPLES**

40071 None.

40072 **APPLICATION USAGE**

40073 On error, the expressions  $(math\_errhandling$  & MATH\_ERRNO) and  $(math\_errhandling$  &  
40074 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

40075 **RATIONALE**

40076 None.

40077 **FUTURE DIRECTIONS**

40078 None.

40079 **SEE ALSO**

40080 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

40081 XBD Section 4.19 (on page 116), <math.h>

40082 **CHANGE HISTORY**

40083 First released in Issue 3.

40084 **Issue 5**

40085 The DESCRIPTION is updated to indicate how an application should check for an error. This  
40086 text was previously published in the APPLICATION USAGE section.

40087 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

40088 **Issue 6**

40089 The *lgamma()* function is no longer marked as an extension.

40090 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999  
40091 standard.

40092 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
40093 revised to align with the ISO/IEC 9899:1999 standard.

40094 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
40095 marked.

40096 Functionality relating to the XSI option is marked.

40097 **Issue 7**

40098 Austin Group Interpretation 1003.1-2001 #156 is applied.

40099 The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan,  $-\text{Inf}$ , or a negative  
40100 integer.

**link()**40101 **NAME**40102 `link, linkat` — link one file to another file relative to two directory file descriptors40103 **SYNOPSIS**

```
40104 #include <unistd.h>
40105 int link(const char *path1, const char *path2);
40106 int linkat(int fd1, const char *path1, int fd2, const char *path2,
40107           int flag);
```

40108 **DESCRIPTION**40109 The `link()` function shall create a new link (directory entry) for the existing file, `path1`.

40110 The `path1` argument points to a pathname naming an existing file. The `path2` argument points to a pathname naming the new directory entry to be created. The `link()` function shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

40113 If `path1` names a directory, `link()` shall fail unless the process has appropriate privileges and the implementation supports using `link()` on directories.

40115 If `path1` names a symbolic link, it is implementation-defined whether `link()` follows the symbolic link, or creates a new link to the symbolic link itself.

40117 Upon successful completion, `link()` shall mark for update the last file status change timestamp of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

40120 If `link()` fails, no link shall be created and the link count of the file shall remain unchanged.

40121 The implementation may require that the calling process has permission to access the existing file.

40123 The `linkat()` function shall be equivalent to the `link()` function except in the case where either `path1` or `path2` or both are relative paths. In this case a relative path `path1` is interpreted relative to the directory associated with the file descriptor `fd1` instead of the current working directory and similarly for `path2` and the file descriptor `fd2`. If the file descriptor was opened without `O_SEARCH`, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function shall not perform the check.

40130 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

40132 `AT_SYMLINK_FOLLOW`40133 If `path1` names a symbolic link, a new link for the target of the symbolic link is created.

40134 If `linkat()` is passed the special value `AT_FDCWD` in the `fd1` or `fd2` parameter, the current working directory is used for the respective `path` argument. If both `fd1` and `fd2` have value `AT_FDCWD`, the behavior shall be identical to a call to `link()`.

40137 If the `AT_SYMLINK_FOLLOW` flag is clear in the `flag` argument and the `path1` argument names a symbolic link, a new link is created for the symbolic link `path1` and not its target.

40139 **RETURN VALUE**

40140 Upon successful completion, these functions shall return 0. Otherwise, these functions shall return `-1` and set `errno` to indicate the error.

40142 **ERRORS**

40143 These functions shall fail if:

40144 [EACCES] A component of either path prefix denies search permission, or the requested  
 40145 link requires writing in a directory that denies write permission, or the calling  
 40146 process does not have permission to access the existing file and this is required  
 40147 by the implementation.

40148 [EEXIST] The *path2* argument resolves to an existing directory entry or refers to a  
 40149 symbolic link.

40150 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or  
 40151 *path2* argument.

40152 [EMLINK] The number of links to the file named by *path1* would exceed {LINK\_MAX}.

40153 [ENAMETOOLONG]

40154 The length of a component of a pathname is longer than {NAME\_MAX}.

40155 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does  
 40156 not exist; or *path1* or *path2* points to an empty string.

40157 [ENOSPC] The directory to contain the link cannot be extended.

40158 [ENOTDIR] A component of either path prefix is not a directory, or the *path1* argument  
 40159 contains at least one non-`<slash>` character and ends with one or more trailing  
 40160 `<slash>` characters and the last pathname component names an existing file  
 40161 that is neither a directory nor a symbolic link to a directory.

40162 [EPERM] The file named by *path1* is a directory and either the calling process does not  
 40163 have appropriate privileges or the implementation prohibits using *link()* on  
 40164 directories.

40165 [EROFS] The requested link requires writing in a directory on a read-only file system.

40166 [EXDEV] The link named by *path2* and the file named by *path1* are on different file  
 40167 systems and the implementation does not support links between file systems.

40168 OB XSR [EXDEV] *path1* refers to a named STREAM.

40169 The *linkat()* function shall fail if:

40170 [EBADF] The *path1* or *path2* argument does not specify an absolute path and the *fd1* or  
 40171 *fd2* argument, respectively, is neither AT\_FDCWD nor a valid file descriptor  
 40172 open for reading.

40173 These functions may fail if:

40174 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 40175 resolution of the *path1* or *path2* argument.

40176 [ENAMETOOLONG]

40177 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 40178 symbolic link produced an intermediate result with a length that exceeds  
 40179 {PATH\_MAX}.

**link()**

40180 The *linkat()* function may fail if:

40181 [EINVAL] The value of the *flag* argument is not valid.

40182 [ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*,  
40183 respectively, is neither AT\_FDCWD nor a file descriptor associated with a  
40184 directory.

40185 **EXAMPLES**40186 **Creating a Link to a File**

40187 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating  
40188 a new directory entry named **/modules/pass1**.

```
40189 #include <unistd.h>
40190
40191 char *path1 = "/home/cnd/mod1";
40192 char *path2 = "/modules/pass1";
40193 int status;
40194 ...
40195 status = link (path1, path2);
```

40195 **Creating a Link to a File Within a Program**

40196 In the following program example, the *link()* function links the **/etc/passwd** file (defined as  
40197 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the  
40198 current password file. Then, after removing the current password file (defined as  
40199 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*  
40200 function again.

```
40201 #include <unistd.h>
40202
40203 #define LOCKFILE "/etc/passwd"
40204 #define PASSWDFILE "/etc/passwd"
40205 #define SAVEFILE "/etc/opasswd"
40206 ...
40207 /* Save current password file */
40208 link (PASSWDFILE, SAVEFILE);
40209
40210 /* Remove current password file. */
40211 unlink (PASSWDFILE);
40212
40213 /* Save new password file as current password file. */
40214 link (LOCKFILE, PASSWDFILE);
```

40212 **APPLICATION USAGE**

40213 Some implementations do allow links between file systems.

40214 If *path1* refers to a symbolic link, application developers should use *linkat()* with appropriate  
40215 flags to select whether or not the symbolic link should be resolved.

40216 **RATIONALE**

40217 Linking to a directory is restricted to the superuser in most historical implementations because  
40218 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This  
40219 volume of POSIX.1-2008 continues that philosophy by prohibiting *link()* and *unlink()* from  
40220 doing this. Other functions could do it if the implementor designed such an extension.

40221 Some historical implementations allow linking of files on different file systems. Wording was

40222 added to explicitly allow this optional behavior.

40223 The exception for cross-file system links is intended to apply only to links that are  
40224 programmatically indistinguishable from “hard” links.

40225 The purpose of the *linkat()* function is to link files in directories other than the current working  
40226 directory without exposure to race conditions. Any part of the path of a file could be changed in  
40227 parallel to a call to *link()*, resulting in unspecified behavior. By opening a file descriptor for the  
40228 directory of both the existing file and the target location and using the *linkat()* function it can be  
40229 guaranteed that the both filenames are in the desired directories.

40230 The AT\_SYMLINK\_FOLLOW flag allows for implementing both common behaviors of the  
40231 *link()* function. The POSIX specification requires that if *path1* is a symbolic link, a new link for  
40232 the target of the symbolic link is created. Many systems by default or as an alternative provide a  
40233 mechanism to avoid the implicit symbolic link lookup and create a new link for the symbolic  
40234 link itself.

40235 Earlier versions of this standard specified only the *link()* function, and required it to behave like  
40236 *linkat()* with the AT\_SYMLINK\_FOLLOW flag. However, historical practice from SVR4 and  
40237 Linux kernels had *link()* behaving like *linkat()* with no flags, and many systems that attempted  
40238 to provide a conforming *link()* function did so in a way that was rarely used, and when it was  
40239 used did not conform to the standard (e.g., by not being atomic, or by dereferencing the  
40240 symbolic link incorrectly). Since applications could not rely on *link()* following links in practice,  
40241 the *linkat()* function was added taking a flag to specify the desired behavior for the application.

#### 40242 FUTURE DIRECTIONS

40243 None.

#### 40244 SEE ALSO

40245 *rename()*, *symlink()*, *unlink()*

40246 XBD <fcntl.h>, <unistd.h>

#### 40247 CHANGE HISTORY

40248 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 40249 Issue 6

40250 The following new requirements on POSIX implementations derive from alignment with the  
40251 Single UNIX Specification:

- 40252 • The [ELOOP] mandatory error condition is added.
- 40253 • A second [ENAMETOOLONG] is added as an optional error condition.

40254 The following changes were made to align with the IEEE P1003.1a draft standard:

- 40255 • An explanation is added of the action when *path2* refers to a symbolic link.
- 40256 • The [ELOOP] optional error condition is added.

#### 40257 Issue 7

40258 Austin Group Interpretation 1003.1-2001 #143 is applied.

40259 SD5-XSH-ERN-93 is applied, adding RATIONALE.

40260 The *linkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set  
40261 Part 2.

40262 Functionality relating to XSI STREAMS is marked obsolescent.

40263 Changes are made related to support for finegrained timestamps.

**link()**

40264

The [EOPNOTSUPP] error is removed.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

40265 **NAME**

40266       lio\_listio — list directed I/O

40267 **SYNOPSIS**

40268       #include &lt;aio.h&gt;

40269       int lio\_listio(int mode, struct aiocb \*restrict const list[restrict],  
40270                    int nent, struct sigevent \*restrict sig);40271 **DESCRIPTION**40272       The *lio\_listio()* function shall initiate a list of I/O requests with a single function call.40273       The *mode* argument takes one of the values LIO\_WAIT or LIO\_NOWAIT declared in <aio.h> and  
40274       determines whether the function returns when the I/O operations have been completed, or as  
40275       soon as the operations have been queued. If the *mode* argument is LIO\_WAIT, the function shall  
40276       wait until all I/O is complete and the *sig* argument shall be ignored.40277       If the *mode* argument is LIO\_NOWAIT, the function shall return immediately, and asynchronous  
40278       notification shall occur, according to the *sig* argument, when all the I/O operations complete. If  
40279       *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous  
40280       notification occurs as specified in Section 2.4.1 (on page 484) when all the requests in *list* have  
40281       completed.40282       The I/O requests enumerated by *list* are submitted in an unspecified order.40283       The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.  
40284       The array may contain NULL elements, which shall be ignored.40285       If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list*  
40286       become illegal addresses before all asynchronous I/O completed and, if necessary, the  
40287       notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio\_buf*  
40288       member of the **aiocb** structure pointed to by the elements of the array *list* become illegal  
40289       addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed,  
40290       the behavior is undefined.40291       The *aio\_lio\_opcode* field of each **aiocb** structure specifies the operation to be performed. The  
40292       supported operations are LIO\_READ, LIO\_WRITE, and LIO\_NOP; these symbols are defined in  
40293       <aio.h>. The LIO\_NOP operation causes the list entry to be ignored. If the *aio\_lio\_opcode*  
40294       element is equal to LIO\_READ, then an I/O operation is submitted as if by a call to *aio\_read()*  
40295       with the *aio\_cbp* equal to the address of the **aiocb** structure. If the *aio\_lio\_opcode* element is equal to  
40296       LIO\_WRITE, then an I/O operation is submitted as if by a call to *aio\_write()* with the *aio\_cbp*  
40297       equal to the address of the **aiocb** structure.40298       The *aio\_fildes* member specifies the file descriptor on which the operation is to be performed.40299       The *aio\_buf* member specifies the address of the buffer to or from which the data is transferred.40300       The *aio\_nbytes* member specifies the number of bytes of data to be transferred.40301       The members of the **aiocb** structure further describe the I/O operation to be performed, in a  
40302       manner identical to that of the corresponding **aiocb** structure when used by the *aio\_read()* and  
40303       *aio\_write()* functions.40304       The *nent* argument specifies how many elements are members of the list; that is, the length of the  
40305       array.40306       The behavior of this function is altered according to the definitions of synchronized I/O data  
40307       integrity completion and synchronized I/O file integrity completion if synchronized I/O is  
40308       enabled on the file associated with *aio\_fildes*.

**lio\_listio()**

40309 For regular files, no data transfer shall occur past the offset maximum established in the open  
40310 file description associated with *aiocbp*→*aio\_fildes*.

40311 If *sig*→*sigev\_notify* is SIGEV\_THREAD and *sig*→*sigev\_notify\_attributes* is a non-null pointer and  
40312 the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O  
40313 being completed, then the behavior is undefined.

**RETURN VALUE**

40314 If the *mode* argument has the value LIO\_NOWAIT, the *lio\_listio()* function shall return the value  
40315 zero if the I/O operations are successfully queued; otherwise, the function shall return the value  
40316 -1 and set *errno* to indicate the error.  
40317

40318 If the *mode* argument has the value LIO\_WAIT, the *lio\_listio()* function shall return the value zero  
40319 when all the indicated I/O has completed successfully. Otherwise, *lio\_listio()* shall return a value  
40320 of -1 and set *errno* to indicate the error.

40321 In either case, the return value only indicates the success or failure of the *lio\_listio()* call itself,  
40322 not the status of the individual I/O requests. In some cases one or more of the I/O requests  
40323 contained in the list may fail. Failure of an individual request does not prevent completion of  
40324 any other individual request. To determine the outcome of each I/O request, the application  
40325 shall examine the error status associated with each **aiocb** control block. The error statuses so  
40326 returned are identical to those returned as the result of an *aio\_read()* or *aio\_write()* function.

**ERRORS**

40327 The *lio\_listio()* function shall fail if:

40328 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The  
40329 application may check the error status for each **aiocb** to determine the  
40330 individual request(s) that failed.  
40331

40332 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit  
40333 {AIO\_MAX} to be exceeded.

40334 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than  
40335 {AIO\_LISTIO\_MAX}.

40336 [EINTR] A signal was delivered while waiting for all I/O requests to complete during  
40337 an LIO\_WAIT operation. Note that, since each I/O operation invoked by  
40338 *lio\_listio()* may possibly provoke a signal when it completes, this error return  
40339 may be caused by the completion of one (or more) of the very I/O operations  
40340 being awaited. Outstanding I/O requests are not canceled, and the application  
40341 shall examine each list element to determine whether the request was  
40342 initiated, canceled, or completed.

40343 [EIO] One or more of the individual I/O operations failed. The application may  
40344 check the error status for each **aiocb** structure to determine the individual  
40345 request(s) that failed.

40346 In addition to the errors returned by the *lio\_listio()* function, if the *lio\_listio()* function succeeds  
40347 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list  
40348 may have been initiated. If the *lio\_listio()* function fails with an error code other than [EAGAIN],  
40349 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation  
40350 indicated by each list element can encounter errors specific to the individual read or write  
40351 function being performed. In this event, the error status for each **aiocb** control block contains the  
40352 associated error code. The error codes that can be set are the same as would be set by a *read()* or  
40353 *write()* function, with the following additional error codes possible:

- 40354 [EAGAIN] The requested I/O operation was not queued due to resource limitations.
- 40355 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
40356 *aio\_cancel()* request.
- 40357 [EFBIG] The *aioctx->aio\_lio\_opcode* is LIO\_WRITE, the file is a regular file,  
40358 *aioctx->aio\_nbytes* is greater than 0, and the *aioctx->aio\_offset* is greater than or  
40359 equal to the offset maximum in the open file description associated with  
40360 *aioctx->aio\_fildes*.
- 40361 [EINPROGRESS] The requested I/O is in progress.
- 40362 [EOVERFLOW] The *aioctx->aio\_lio\_opcode* is LIO\_READ, the file is a regular file,  
40363 *aioctx->aio\_nbytes* is greater than 0, and the *aioctx->aio\_offset* is before the  
40364 end-of-file and is greater than or equal to the offset maximum in the open file  
40365 description associated with *aioctx->aio\_fildes*.

**EXAMPLES**

40366 None.

**APPLICATION USAGE**

40368 None.

**RATIONALE**

40371 Although it may appear that there are inconsistencies in the specified circumstances for error  
40372 codes, the [EIO] error condition applies when any circumstance relating to an individual  
40373 operation makes that operation fail. This might be due to a badly formulated request (for  
40374 example, the *aio\_lio\_opcode* field is invalid, and *aio\_error()* returns [EINVAL]) or might arise from  
40375 application behavior (for example, the file descriptor is closed before the operation is initiated,  
40376 and *aio\_error()* returns [EBADF]).

40377 The limitation on the set of error codes returned when operations from the list shall have been  
40378 initiated enables applications to know when operations have been started and whether  
40379 *aio\_error()* is valid for a specific operation.

**FUTURE DIRECTIONS**

40380 None.

**SEE ALSO**

40383 *aio\_read()*, *aio\_write()*, *aio\_error()*, *aio\_return()*, *aio\_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,  
40384 *read()*

40385 XBD <[aio.h](#)>

**CHANGE HISTORY**

40387 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

40388 Large File Summit extensions are added.

**Issue 6**

40390 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
40391 implementation does not support the Asynchronous Input and Output option.

40392 The *lio\_listio()* function is marked as part of the Asynchronous Input and Output option.

40393 The following new requirements on POSIX implementations derive from alignment with the  
40394 Single UNIX Specification:

**lio\_listio()**

- 40395 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs  
40396 past the offset maximum established in the open file description associated with  
40397 *aiocbp*→*aio\_fildes*. This change is to support large files.
- 40398 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support  
40399 large files.
- 40400 The normative text is updated to avoid use of the term “must” for application requirements.
- 40401 The **restrict** keyword is added to the *lio\_listio()* prototype for alignment with the  
40402 ISO/IEC 9899: 1999 standard.
- 40403 **Issue 6**
- 40404 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for  
40405 symmetry with the *aio\_read()* and *aio\_write()* functions to the DESCRIPTION.
- 40406 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the  
40407 DESCRIPTION making it explicit that the user is required to keep the structure pointed to by  
40408 *sig*→*sigev\_notify\_attributes* valid until the last asynchronous operation finished and the  
40409 notification has been sent.
- 40410 **Issue 7**
- 40411 The *lio\_listio()* function is moved from the Asynchronous Input and Output option to the Base.

40412 **NAME**

40413 listen — listen for socket connections and limit the queue of incoming connections

40414 **SYNOPSIS**

```
40415 #include <sys/socket.h>
40416 int listen(int socket, int backlog);
```

40417 **DESCRIPTION**

40418 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as  
 40419 accepting connections.

40420 The *backlog* argument provides a hint to the implementation which the implementation shall use  
 40421 to limit the number of outstanding connections in the socket's listen queue. Implementations  
 40422 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*  
 40423 argument value shall result in a larger or equal length of the listen queue. Implementations shall  
 40424 support values of *backlog* up to SOMAXCONN, defined in *<sys/socket.h>*.

40425 The implementation may include incomplete connections in its listen queue. The limits on the  
 40426 number of incomplete connections and completed connections queued may be different.

40427 The implementation may have an upper limit on the length of the listen queue—either global or  
 40428 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

40429 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it  
 40430 had been called with a *backlog* argument value of 0.

40431 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of  
 40432 the listen queue may be set to an implementation-defined minimum value.

40433 The socket in use may require the process to have appropriate privileges to use the *listen()*  
 40434 function.

40435 **RETURN VALUE**

40436 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set  
 40437 to indicate the error.

40438 **ERRORS**

40439 The *listen()* function shall fail if:

40440 [EBADF] The *socket* argument is not a valid file descriptor.

40441 [EDESTADDRREQ]

40442 The socket is not bound to a local address, and the protocol does not support  
 40443 listening on an unbound socket.

40444 [EINVAL] The *socket* is already connected.

40445 [ENOTSOCK] The *socket* argument does not refer to a socket.

40446 [EOPNOTSUPP] The socket protocol does not support *listen()*.

40447 The *listen()* function may fail if:

40448 [EACCES] The calling process does not have appropriate privileges.

40449 [EINVAL] The *socket* has been shut down.

40450 [ENOBUFS] Insufficient resources are available in the system to complete the call.

**listen()**

System Interfaces

40451 **EXAMPLES**

40452 None.

40453 **APPLICATION USAGE**

40454 None.

40455 **RATIONALE**

40456 None.

40457 **FUTURE DIRECTIONS**

40458 None.

40459 **SEE ALSO**40460 *accept()*, *connect()*, *socket()*

40461 XBD &lt;sys/socket.h&gt;

40462 **CHANGE HISTORY**

40463 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40464 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*

40465 argument.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***llabs()**40466 **NAME**

40467 llabs — return a long integer absolute value

40468 **SYNOPSIS**

40469 #include &lt;stdlib.h&gt;

40470 long long llabs(long long i);

40471 **DESCRIPTION**40472 Refer to *labs()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**lldiv()**40473 **NAME**40474 `lldiv` — compute quotient and remainder of a long division40475 **SYNOPSIS**40476 `#include <stdlib.h>`40477 `lldiv_t lldiv(long long numer, long long denom);`40478 **DESCRIPTION**40479 Refer to *ldiv()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

40480 **NAME**

40481 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

40482 **SYNOPSIS**

```
40483 #include <math.h>
40484 long long llrint(double x);
40485 long long llrintf(float x);
40486 long long llrintl(long double x);
```

40487 **DESCRIPTION**

40488 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40489 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40490 volume of POSIX.1-2008 defers to the ISO C standard.

40491 These functions shall round their argument to the nearest integer value, rounding according to  
 40492 the current rounding direction.

40493 An application wishing to check for error situations should set *errno* to zero and call  
 40494 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 40495 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 40496 zero, an error has occurred.

40497 **RETURN VALUE**

40498 Upon successful completion, these functions shall return the rounded integer value.

40499 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

40500 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

40501 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

40502 If the correct value is positive and too large to represent as a **long long**, an unspecified value  
 40503 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 40504 shall occur;

40505 CX otherwise, a domain error may occur.

40506 If the correct value is negative and too large to represent as a **long long**, an unspecified value  
 40507 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 40508 shall occur;

40509 CX otherwise, a domain error may occur.

40510 **ERRORS**

40511 These functions shall fail if:

40512 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 40513 integer.

40514 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40515 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 40516 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 40517 shall be raised.

40518 These functions may fail if:

40519 **Domain Error** The correct value is not representable as an integer.

40520 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40521 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 40522 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 40523 shall be raised.

## llrint()

### 40524 EXAMPLES

40525 None.

### 40526 APPLICATION USAGE

40527 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
40528 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

### 40529 RATIONALE

40530 These functions provide floating-to-integer conversions. They round according to the current  
40531 rounding direction. If the rounded value is outside the range of the return type, the numeric  
40532 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
40533 floating-point exception and the result differs from the argument, they raise the inexact floating-  
40534 point exception.

### 40535 FUTURE DIRECTIONS

40536 None.

### 40537 SEE ALSO

40538 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [lrint\(\)](#)

40539 XBD Section 4.19 (on page 116), [<math.h>](#)

### 40540 CHANGE HISTORY

40541 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

### 40542 Issue 7

40543 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 is applied.

40544 **NAME**

40545 llround, llroundf, llroundl — round to nearest integer value

40546 **SYNOPSIS**

```
40547 #include <math.h>
40548 long long llround(double x);
40549 long long llroundf(float x);
40550 long long llroundl(long double x);
```

40551 **DESCRIPTION**

40552 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40553 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40554 volume of POSIX.1-2008 defers to the ISO C standard.

40555 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 40556 away from zero, regardless of the current rounding direction.

40557 An application wishing to check for error situations should set *errno* to zero and call  
 40558 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 40559 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 40560 zero, an error has occurred.

40561 **RETURN VALUE**

40562 Upon successful completion, these functions shall return the rounded integer value.

40563 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

40564 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

40565 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

40566 If the correct value is positive and too large to represent as a **long long**, an unspecified value  
 40567 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 40568 shall occur;

40569 CX otherwise, a domain error may occur.

40570 If the correct value is negative and too large to represent as a **long long**, an unspecified value  
 40571 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 40572 shall occur;

40573 CX otherwise, a domain error may occur.

40574 **ERRORS**

40575 These functions shall fail if:

40576 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 40577 integer.

40578 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40579 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 40580 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 40581 shall be raised.

40582 These functions may fail if:

40583 **Domain Error** The correct value is not representable as an integer.

40584 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 40585 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 40586 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 40587 shall be raised.

**llround()**40588 **EXAMPLES**

40589 None.

40590 **APPLICATION USAGE**40591 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
40592 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.40593 **RATIONALE**40594 These functions differ from the *llrint()* functions in that the default rounding direction for the  
40595 *llround()* functions round halfway cases away from zero and need not raise the inexact floating-  
40596 point exception for non-integer arguments that round to within the range of the return type.40597 **FUTURE DIRECTIONS**

40598 None.

40599 **SEE ALSO**40600 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [lround\(\)](#)40601 XBD Section 4.19 (on page 116), [<math.h>](#)40602 **CHANGE HISTORY**

40603 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

40604 **Issue 7**

40605 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

40606 **NAME**

40607 localeconv — return locale-specific information

40608 **SYNOPSIS**

40609 #include &lt;locale.h&gt;

40610 struct lconv \*localeconv(void);

40611 **DESCRIPTION**

40612 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40613 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40614 volume of POSIX.1-2008 defers to the ISO C standard.

40615 The *localeconv()* function shall set the components of an object with the type **struct lconv** with  
 40616 the values appropriate for the formatting of numeric quantities (monetary and otherwise)  
 40617 according to the rules of the current locale.

40618 The members of the structure with type **char \*** are pointers to strings, any of which (except  
 40619 **decimal\_point**) can point to "", to indicate that the value is not available in the current locale or  
 40620 is of zero length. The members with type **char** are non-negative numbers, any of which can be  
 40621 {CHAR\_MAX} to indicate that the value is not available in the current locale.

40622 The members include the following:

40623 **char \*decimal\_point**

40624 The radix character used to format non-monetary quantities.

40625 **char \*thousands\_sep**

40626 The character used to separate groups of digits before the decimal-point character in  
 40627 formatted non-monetary quantities.

40628 **char \*grouping**

40629 A string whose elements taken as one-byte integer values indicate the size of each group of  
 40630 digits in formatted non-monetary quantities.

40631 **char \*int\_curr\_symbol**

40632 The international currency symbol applicable to the current locale. The first three characters  
 40633 contain the alphabetic international currency symbol in accordance with those specified in  
 40634 the ISO 4217:2001 standard. The fourth character (immediately preceding the null byte) is  
 40635 the character used to separate the international currency symbol from the monetary  
 40636 quantity.

40637 **char \*currency\_symbol**

40638 The local currency symbol applicable to the current locale.

40639 **char \*mon\_decimal\_point**

40640 The radix character used to format monetary quantities.

40641 **char \*mon\_thousands\_sep**

40642 The separator for groups of digits before the decimal-point in formatted monetary  
 40643 quantities.

40644 **char \*mon\_grouping**

40645 A string whose elements taken as one-byte integer values indicate the size of each group of  
 40646 digits in formatted monetary quantities.

40647 **char \*positive\_sign**

40648 The string used to indicate a non-negative valued formatted monetary quantity.

**localeconv()**

40649	<b>char *negative_sign</b>
40650	The string used to indicate a negative valued formatted monetary quantity.
40651	<b>char int_frac_digits</b>
40652	The number of fractional digits (those after the decimal-point) to be displayed in an
40653	internationally formatted monetary quantity.
40654	<b>char frac_digits</b>
40655	The number of fractional digits (those after the decimal-point) to be displayed in a
40656	formatted monetary quantity.
40657	<b>char p_cs_precedes</b>
40658	Set to 1 if the <b>currency_symbol</b> precedes the value for a non-negative formatted monetary
40659	quantity. Set to 0 if the symbol succeeds the value.
40660	<b>char p_sep_by_space</b>
40661	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the
40662	value for a non-negative formatted monetary quantity.
40663	<b>char n_cs_precedes</b>
40664	Set to 1 if the <b>currency_symbol</b> precedes the value for a negative formatted monetary
40665	quantity. Set to 0 if the symbol succeeds the value.
40666	<b>char n_sep_by_space</b>
40667	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the
40668	value for a negative formatted monetary quantity.
40669	<b>char p_sign_posn</b>
40670	Set to a value indicating the positioning of the <b>positive_sign</b> for a non-negative formatted
40671	monetary quantity.
40672	<b>char n_sign_posn</b>
40673	Set to a value indicating the positioning of the <b>negative_sign</b> for a negative formatted
40674	monetary quantity.
40675	<b>char int_p_cs_precedes</b>
40676	Set to 1 or 0 if the <b>int_curr_symbol</b> respectively precedes or succeeds the value for a non-
40677	negative internationally formatted monetary quantity.
40678	<b>char int_n_cs_precedes</b>
40679	Set to 1 or 0 if the <b>int_curr_symbol</b> respectively precedes or succeeds the value for a
40680	negative internationally formatted monetary quantity.
40681	<b>char int_p_sep_by_space</b>
40682	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the
40683	value for a non-negative internationally formatted monetary quantity.
40684	<b>char int_n_sep_by_space</b>
40685	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the
40686	value for a negative internationally formatted monetary quantity.
40687	<b>char int_p_sign_posn</b>
40688	Set to a value indicating the positioning of the <b>positive_sign</b> for a non-negative
40689	internationally formatted monetary quantity.
40690	<b>char int_n_sign_posn</b>
40691	Set to a value indicating the positioning of the <b>negative_sign</b> for a negative internationally
40692	formatted monetary quantity.

- 40693 The elements of **grouping** and **mon\_grouping** are interpreted according to the following:
- 40694 {CHAR\_MAX} No further grouping is to be performed.
- 40695 0 The previous element is to be repeatedly used for the remainder of the digits.
- 40696 *other* The integer value is the number of digits that comprise the current group. The  
40697 next element is examined to determine the size of the next group of digits  
40698 before the current group.
- 40699 The values of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space**  
40700 are interpreted according to the following:
- 40701 0 No space separates the currency symbol and value.
- 40702 1 If the currency symbol and sign string are adjacent, a space separates them from the value;  
40703 otherwise, a space separates the currency symbol from the value.
- 40704 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a  
40705 space separates the sign string from the value.
- 40706 For **int\_p\_sep\_by\_space** and **int\_n\_sep\_by\_space**, the fourth character of **int\_curr\_symbol** is  
40707 used instead of a space.
- 40708 The values of **p\_sign\_posn**, **n\_sign\_posn**, **int\_p\_sign\_posn**, and **int\_n\_sign\_posn** are  
40709 interpreted according to the following:
- 40710 0 Parentheses surround the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 40711 1 The sign string precedes the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 40712 2 The sign string succeeds the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 40713 3 The sign string immediately precedes the **currency\_symbol** or **int\_curr\_symbol**.
- 40714 4 The sign string immediately succeeds the **currency\_symbol** or **int\_curr\_symbol**.
- 40715 The implementation shall behave as if no function in this volume of POSIX.1-2008 calls  
40716 *localeconv()*.
- 40717 CX **The *localeconv()* function need not be thread-safe.**
- 40718 **RETURN VALUE**
- 40719 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not  
40720 modify the structure pointed to by the return value which may be overwritten by a subsequent  
40721 call to *localeconv()*. In addition, calls to *setlocale()* with the categories *LC\_ALL*, *LC\_MONETARY*,  
40722 or *LC\_NUMERIC* or calls to *uselocale()* which change the categories *LC\_MONETARY* or  
40723 *LC\_NUMERIC* may overwrite the contents of the structure.
- 40724 **ERRORS**
- 40725 No errors are defined.

# localeconv()

40726 **EXAMPLES**

40727 None.

40728 **APPLICATION USAGE**

40729 The following table illustrates the rules which may be used by four countries to format  
40730 monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	€1.230	-€1.230	EUR.1.230
Netherlands	€ 1.234,56	€ -1.234,56	EUR 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

40736 For these four countries, the respective values for the monetary members of the structure  
40737 returned by *localeconv()* are:

	Italy	Netherlands	Norway	Switzerland
40738 <b>int_curr_symbol</b>	"EUR. "	"EUR "	"NOK "	"CHF "
40739 <b>currency_symbol</b>	"€. "	"€"	"kr"	"SFrs. "
40740 <b>mon_decimal_point</b>	" "	", "	", "	". "
40741 <b>mon_thousands_sep</b>	". "	". "	". "	", "
40742 <b>mon_grouping</b>	"\3"	"\3"	"\3"	"\3"
40743 <b>positive_sign</b>	" "	" "	" "	" "
40744 <b>negative_sign</b>	"-"	" "	"-"	"C"
40745 <b>int_frac_digits</b>	0	2	2	2
40746 <b>frac_digits</b>	0	2	2	2
40747 <b>p_cs_precedes</b>	1	1	1	1
40748 <b>p_sep_by_space</b>	0	1	0	0
40749 <b>n_cs_precedes</b>	1	1	1	1
40750 <b>n_sep_by_space</b>	0	1	0	0
40751 <b>p_sign_posn</b>	1	1	1	1
40752 <b>n_sign_posn</b>	1	4	2	2
40753 <b>int_p_cs_precedes</b>	1	1	1	1
40754 <b>int_n_cs_precedes</b>	1	1	1	1
40755 <b>int_p_sep_by_space</b>	0	0	0	0
40756 <b>int_n_sep_by_space</b>	0	0	0	0
40757 <b>int_p_sign_posn</b>	1	1	1	1
40758 <b>int_n_sign_posn</b>	1	4	4	2

40760 **RATIONALE**

40761 None.

40762 **FUTURE DIRECTIONS**

40763 None.

40764 **SEE ALSO**

40765 *fprintf()*, *fscanf()*, *isalpha()*, *isascii()*, *nl\_langinfo()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,  
40766 *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, *uselocale()*

40767 XBD [<langinfo.h>](#), [<locale.h>](#)

40768 **CHANGE HISTORY**

40769 First released in Issue 4. Derived from the ANSI C standard.

40770 **Issue 6**

40771 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

40772 The RETURN VALUE section is rewritten to avoid use of the term “must”.

40773 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard.

40774 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

40775 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int\_curr\_symbol** and updating the descriptions of **p\_sep\_by\_space** and **n\_sep\_by\_space**. These changes are for alignment with the ISO C standard.

40778 **Issue 7**

40779 Austin Group Interpretation 1003.1-2001 #156 is applied.

40780 The definitions of **int\_curr\_symbol** and **currency\_symbol** are updated.

40781 The examples in the APPLICATION USAGE section are updated.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**localtime()**40782 **NAME**40783 `localtime, localtime_r` — convert a time value to a broken-down local time40784 **SYNOPSIS**40785 `#include <time.h>`40786 `struct tm *localtime(const time_t *timer);`40787 CX `struct tm *localtime_r(const time_t *restrict timer,`  
40788 `struct tm *restrict result);`40789 **DESCRIPTION**40790 CX For `localtime()`: The functionality described on this reference page is aligned with the ISO C  
40791 standard. Any conflict between the requirements described here and the ISO C standard is  
40792 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.40793 The `localtime()` function shall convert the time in seconds since the Epoch pointed to by `timer`  
40794 into a broken-down time, expressed as a local time. The function corrects for the timezone and  
40795 CX any seasonal time adjustments. Local timezone information is used as though `localtime()` calls  
40796 `tzset()`.40797 The relationship between a time in seconds since the Epoch used as an argument to `localtime()`  
40798 and the `tm` structure (defined in the `<time.h>` header) is that the result shall be as specified in  
40799 the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page  
40800 113) corrected for timezone and any seasonal time adjustments, where the names in the structure  
40801 and in the expression correspond.40802 The same relationship shall apply for `localtime_r()`.40803 The `localtime()` function need not be thread-safe.40804 The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static  
40805 objects: a broken-down time structure and an array of type `char`. Execution of any of the  
40806 functions may overwrite the information returned in either of these objects by any of the other  
40807 functions.40808 The `localtime_r()` function shall convert the time in seconds since the Epoch pointed to by `timer`  
40809 into a broken-down time stored in the structure to which `result` points. The `localtime_r()` function  
40810 shall also return a pointer to that same structure.40811 Unlike `localtime()`, the `localtime_r()` function is not required to set `tzname`. If `localtime_r()` does  
40812 not set `tzname`, it shall not set `daylight` and shall not set `timezone`.40813 **RETURN VALUE**40814 Upon successful completion, the `localtime()` function shall return a pointer to the broken-down  
40815 CX time structure. If an error is detected, `localtime()` shall return a null pointer and set `errno` to  
40816 indicate the error.40817 Upon successful completion, `localtime_r()` shall return a pointer to the structure pointed to by the  
40818 argument `result`. If an error is detected, `localtime_r()` shall return a null pointer and set `errno` to  
40819 indicate the error.40820 **ERRORS**40821 CX The `localtime()` and `localtime_r()` functions shall fail if:

40822 CX [EOVERFLOW] The result cannot be represented.

## 40823 EXAMPLES

40824 **Getting the Local Date and Time**

40825 The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 40826 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,  
 40827 and *asctime()* to convert the broken-down time values into a printable string.

```
40828 #include <stdio.h>
40829 #include <time.h>

40830 int main(void)
40831 {
40832     time_t result;

40833     result = time(NULL);
40834     printf("%s%ju secs since the Epoch\n",
40835           asctime(localtime(&result)),
40836           (uintmax_t)result);
40837     return(0);
40838 }
```

40839 This example writes the current time to *stdout* in a form like this:

```
40840 Wed Jun 26 10:32:15 1996
40841 835810335 secs since the Epoch
```

40842 **Getting the Modification Time for a File**

40843 The following example prints the last data modification timestamp in the local timezone for a  
 40844 given file.

```
40845 #include <stdio.h>
40846 #include <time.h>
40847 #include <sys/stat.h>

40848 int
40849 print_file_time(const char *filename)
40850 {
40851     struct stat statbuf;
40852     struct tm *tm;
40853     char timestr[BUFSIZ];

40854     if(stat(filename, &statbuf) == -1)
40855         return -1;
40856     if((tm = localtime(&statbuf.st_mtime)) == NULL)
40857         return -1;
40858     if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
40859         return -1;
40860     printf("%s: %s.%09ld\n", filename, timestr, statbuf.st_mtim.tv_nsec);
40861     return 0;
40862 }
```

**localtime()**40863 **Timing an Event**

40864 The following example gets the current time, converts it to a string using *localtime()* and  
 40865 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 40866 an event being timed.

```
40867 #include <time.h>
40868 #include <stdio.h>
40869 ...
40870 time_t now;
40871 int minutes_to_event;
40872 ...
40873 time(&now);
40874 printf("The time is ");
40875 fputs(asctime(localtime(&now)), stdout);
40876 printf("There are still %d minutes to the event.\n",
40877     minutes_to_event);
40878 ...
```

40879 **APPLICATION USAGE**

40880 The *localtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 40881 possibly using a static data area that may be overwritten by each call.

40882 **RATIONALE**

40883 None.

40884 **FUTURE DIRECTIONS**

40885 None.

40886 **SEE ALSO**

40887 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*,  
 40888 *tzset()*, *utime()*

40889 XBD Section 4.15 (on page 113), [time.h](#)

40890 **CHANGE HISTORY**

40891 First released in Issue 1. Derived from Issue 1 of the SVID.

40892 **Issue 5**

40893 A note indicating that the *localtime()* function need not be reentrant is added to the  
 40894 DESCRIPTION.

40895 The *localtime\_r()* function is included for alignment with the POSIX Threads Extension.

40896 **Issue 6**

40897 The *localtime\_r()* function is marked as part of the Thread-Safe Functions option.

40898 Extensions beyond the ISO C standard are marked.

40899 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 40900 its avoidance of possibly using a static data area.

40901 The **restrict** keyword is added to the *localtime\_r()* prototype for alignment with the  
 40902 ISO/IEC 9899:1999 standard.

40903 Examples are added.

40904 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the  
 40905 [EOverflow] error.

- 40906 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling  
40907 for *localtime\_r()*.
- 40908 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if  
40909 *localtime\_r()* does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On  
40910 systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide  
40911 information for the same timezone. This updates the description of *localtime\_r()* to mention  
40912 *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.
- 40913 **Issue 7**
- 40914 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 40915 The *localtime\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 40916 Changes are made to the EXAMPLES section related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**lockf()**40917 **NAME**

40918 lockf — record locking on files

40919 **SYNOPSIS**

```
40920 XSI #include <unistd.h>
40921 int lockf(int fildev, int function, off_t size);
```

40922 **DESCRIPTION**

40923 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from  
 40924 threads in other processes which attempt to lock the locked file section shall either return an  
 40925 error value or block until the section becomes unlocked. All the locks for a process are removed  
 40926 when the process terminates. Record locking with *lockf()* shall be supported for regular files and  
 40927 may be supported for other files.

40928 The *fildev* argument is an open file descriptor. To establish a lock with this function, the file  
 40929 descriptor shall be opened with write-only permission (O\_WRONLY) or with read/write  
 40930 permission (O\_RDWR).

40931 The *function* argument is a control value which specifies the action to be taken. The permissible  
 40932 values for *function* are defined in <unistd.h> as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

40933 F\_TEST shall detect if a lock by another process is present on the specified section.

40934 F\_LOCK and F\_TLOCK shall both lock a section of a file if the section is available.

40935 F\_ULOCK shall remove locks from a section of the file.

40936 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be  
 40937 locked or unlocked starts at the current offset in the file and extends forward for a positive size  
 40938 or backward for a negative size (the preceding bytes up to but not including the current offset).  
 40939 If *size* is 0, the section from the current offset through the largest possible file offset shall be  
 40940 locked (that is, from the current offset through the present or any future end-of-file). An area  
 40941 need not be allocated to the file to be locked because locks may exist past the end-of-file.

40942 The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained  
 40943 by a previously locked section for the same process. When this occurs, or if adjacent locked  
 40944 sections would occur, the sections shall be combined into a single locked section. If the request  
 40945 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

40946 F\_LOCK and F\_TLOCK requests differ only by the action taken if the section is not available.  
 40947 F\_LOCK shall block the calling thread until the section is available. F\_TLOCK shall cause the  
 40948 function to fail if the section is already locked by another process.

40949 File locks shall be released on first close by the locking process of any file descriptor for the file.

40950 F\_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the  
 40951 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or  
 40952 to the end-of-file if *size* is (off\_t)0. When all of a locked section is not released (that is, when the  
 40953 beginning or end of the area to be unlocked falls within a locked section), the remaining portions  
 40954 of that section shall remain locked by the process. Releasing the center portion of a locked  
 40955 section shall cause the remaining locked beginning and end portions to become two separate

- 40961 locked sections. If the request would cause the number of locks in the system to exceed a system-  
40962 imposed limit, the request shall fail.
- 40963 A potential for deadlock occurs if the threads of a process controlling a locked section are  
40964 blocked by accessing a locked section of another process. If the system detects that deadlock  
40965 would occur, *lockf()* shall fail with an [EDEADLK] error.
- 40966 The interaction between *fcntl()* and *lockf()* locks is unspecified.
- 40967 Blocking on a section shall be interrupted by any signal.
- 40968 An F\_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested  
40969 section is the maximum value for an object of type `off_t`, when the process has an existing lock  
40970 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a  
40971 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an  
40972 F\_ULOCK request shall attempt to unlock only the requested section.
- 40973 Attempting to lock a section of a file that is associated with a buffered stream produces  
40974 unspecified results.
- 40975 **RETURN VALUE**
- 40976 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to  
40977 indicate an error, and existing locks shall not be changed.
- 40978 **ERRORS**
- 40979 The *lockf()* function shall fail if:
- 40980 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F\_LOCK  
40981 or F\_TLOCK and *fildev* is not a valid file descriptor open for writing.
- 40982 [EACCES] or [EAGAIN]  
40983 The *function* argument is F\_TLOCK or F\_TEST and the section is already  
40984 locked by another process.
- 40985 [EDEADLK] The *function* argument is F\_LOCK and a deadlock is detected.
- 40986 [EINTR] A signal was caught during execution of the function.
- 40987 [EINVAL] The *function* argument is not one of F\_LOCK, F\_TLOCK, F\_TEST, or  
40988 F\_ULOCK; or *size* plus the current file offset is less than 0.
- 40989 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested  
40990 section cannot be represented correctly in an object of type `off_t`.
- 40991 The *lockf()* function may fail if:
- 40992 [EAGAIN] The *function* argument is F\_LOCK or F\_TLOCK and the file is mapped with  
40993 *mmap()*.
- 40994 [EDEADLK] or [ENOLCK]  
40995 The *function* argument is F\_LOCK, F\_TLOCK, or F\_ULOCK, and the request  
40996 would cause the number of locks to exceed a system-imposed limit.
- 40997 [EOPNOTSUPP] or [EINVAL]  
40998 The implementation does not support the locking of files of the type indicated  
40999 by the *fildev* argument.

**lockf()**41000 **EXAMPLES**41001 **Locking a Portion of a File**

41002 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that  
 41003 use locking are prevented from changing it during this process. Only the first 10 000 bytes are  
 41004 locked, and the lock call fails if another process has any part of this area locked already.

```
41005 #include <fcntl.h>
41006 #include <unistd.h>
41007
41008 int fildes;
41009 int status;
41009 ...
41010 fildes = open("/home/cnd/mod1", O_RDWR);
41011 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

41012 **APPLICATION USAGE**

41013 Record-locking should not be used in combination with the `fopen()`, `fread()`, `fwrite()`, and other  
 41014 `stdio` functions. Instead, the more primitive, non-buffered functions (such as `open()`) should be  
 41015 used. Unexpected results may occur in processes that do buffering in the user address space. The  
 41016 process may later read/write data which is/was locked. The `stdio` functions are the most  
 41017 common source of unexpected buffering.

41018 The `alarm()` function may be used to provide a timeout facility in applications requiring it.

41019 **RATIONALE**

41020 None.

41021 **FUTURE DIRECTIONS**

41022 None.

41023 **SEE ALSO**

41024 `alarm()`, `chmod()`, `close()`, `creat()`, `fcntl()`, `fopen()`, `mmap()`, `open()`, `read()`, `write()`

41025 XBD `<unistd.h>`

41026 **CHANGE HISTORY**

41027 First released in Issue 4, Version 2.

41028 **Issue 5**

41029 Moved from X/OPEN UNIX extension to BASE.

41030 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified  
 41031 and moved from optional to mandatory status.

41032 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a  
 41033 file that is associated with a buffered stream.

41034 **Issue 6**

41035 The normative text is updated to avoid use of the term “must” for application requirements.

41036 **Issue 7**

41037 Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

41038 **NAME**

41039 log, logf, logl — natural logarithm function

41040 **SYNOPSIS**

```
41041 #include <math.h>
41042 double log(double x);
41043 float logf(float x);
41044 long double logl(long double x);
```

41045 **DESCRIPTION**

41046 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 41047 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41048 volume of POSIX.1-2008 defers to the ISO C standard.

41049 These functions shall compute the natural logarithm of their argument  $x$ ,  $\log_e(x)$ .

41050 An application wishing to check for error situations should set *errno* to zero and call  
 41051 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41052 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41053 zero, an error has occurred.

41054 **RETURN VALUE**

41055 Upon successful completion, these functions shall return the natural logarithm of  $x$ .

41056 If  $x$  is  $\pm 0$ , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return `-HUGE_VAL`,  
 41057 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

41058 **MX** For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf`, a domain error shall occur, and either a  
 41059 NaN (if supported), or an implementation-defined value shall be returned.

41060 **MX** If  $x$  is NaN, a NaN shall be returned.

41061 If  $x$  is 1, `+0` shall be returned.

41062 If  $x$  is `+Inf`,  $x$  shall be returned.

41063 **ERRORS**

41064 These functions shall fail if:

41065 **MX** Domain Error The finite value of  $x$  is negative, or  $x$  is `-Inf`.

41066 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41067 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41068 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41069 shall be raised.

41070 Pole Error The value of  $x$  is zero.

41071 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41072 then *errno* shall be set to [ERANGE]. If the integer expression  
 41073 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41074 floating-point exception shall be raised.

**log()**41075 **EXAMPLES**

41076 None.

41077 **APPLICATION USAGE**41078 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41079 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41080 **RATIONALE**

41081 None.

41082 **FUTURE DIRECTIONS**

41083 None.

41084 **SEE ALSO**41085 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *log10()*, *log1p()*

41086 XBD Section 4.19 (on page 116), &lt;math.h&gt;

41087 **CHANGE HISTORY**

41088 First released in Issue 1. Derived from Issue 1 of the SVID.

41089 **Issue 5**41090 The DESCRIPTION is updated to indicate how an application should check for an error. This  
41091 text was previously published in the APPLICATION USAGE section.41092 **Issue 6**

41093 The normative text is updated to avoid use of the term “must” for application requirements.

41094 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.41095 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
41096 revised to align with the ISO/IEC 9899:1999 standard.41097 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
41098 marked.

41099 **NAME**

41100 log10, log10f, log10l — base 10 logarithm function

41101 **SYNOPSIS**

```
41102 #include <math.h>
41103 double log10(double x);
41104 float log10f(float x);
41105 long double log10l(long double x);
```

41106 **DESCRIPTION**

41107 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 41108 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41109 volume of POSIX.1-2008 defers to the ISO C standard.

41110 These functions shall compute the base 10 logarithm of their argument  $x$ ,  $\log_{10}(x)$ .

41111 An application wishing to check for error situations should set *errno* to zero and call  
 41112 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41113 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41114 zero, an error has occurred.

41115 **RETURN VALUE**

41116 Upon successful completion, these functions shall return the base 10 logarithm of  $x$ .

41117 If  $x$  is  $\pm 0$ , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return  $-\text{HUGE\_VAL}$ ,  
 41118  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

41119 **MX** For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 41120 NaN (if supported), or an implementation-defined value shall be returned.

41121 **MX** If  $x$  is NaN, a NaN shall be returned.

41122 If  $x$  is 1,  $+0$  shall be returned.

41123 If  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

41124 **ERRORS**

41125 These functions shall fail if:

41126 **MX** Domain Error The finite value of  $x$  is negative, or  $x$  is  $-\text{Inf}$ .

41127 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41128 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41129 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41130 shall be raised.

41131 Pole Error The value of  $x$  is zero.

41132 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41133 then *errno* shall be set to [ERANGE]. If the integer expression  
 41134 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41135 floating-point exception shall be raised.

**log10()**41136 **EXAMPLES**

41137 None.

41138 **APPLICATION USAGE**41139 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41140 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41141 **RATIONALE**

41142 None.

41143 **FUTURE DIRECTIONS**

41144 None.

41145 **SEE ALSO**41146 *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*, *pow()*

41147 XBD Section 4.19 (on page 116), &lt;math.h&gt;

41148 **CHANGE HISTORY**

41149 First released in Issue 1. Derived from Issue 1 of the SVID.

41150 **Issue 5**41151 The DESCRIPTION is updated to indicate how an application should check for an error. This  
41152 text was previously published in the APPLICATION USAGE section.41153 **Issue 6**

41154 The normative text is updated to avoid use of the term “must” for application requirements.

41155 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999  
41156 standard.41157 The DESCRIPTION, RETURN VALUE ERRORS, and APPLICATION USAGE sections are  
41158 revised to align with the ISO/IEC 9899:1999 standard.41159 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
41160 marked.

## 41161 NAME

41162 log1p, log1pf, log1pl — compute a natural logarithm

## 41163 SYNOPSIS

```
41164 #include <math.h>
41165 double log1p(double x);
41166 float log1pf(float x);
41167 long double log1pl(long double x);
```

## 41168 DESCRIPTION

41169 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41170 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41171 volume of POSIX.1-2008 defers to the ISO C standard.

41172 These functions shall compute  $\log_e(1.0 + x)$ .

41173 An application wishing to check for error situations should set *errno* to zero and call  
 41174 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41175 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41176 zero, an error has occurred.

## 41177 RETURN VALUE

41178 Upon successful completion, these functions shall return the natural logarithm of  $1.0 + x$ .

41179 If  $x$  is  $-1$ , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return  $-\text{HUGE\_VAL}$ ,  
 41180  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

41181 MX For finite values of  $x$  that are less than  $-1$ , or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 41182 NaN (if supported), or an implementation-defined value shall be returned.

41183 MX If  $x$  is NaN, a NaN shall be returned.

41184 If  $x$  is  $\pm 0$ , or  $+\text{Inf}$ ,  $x$  shall be returned.

41185 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

## 41186 ERRORS

41187 These functions shall fail if:

41188 MX Domain Error The finite value of  $x$  is less than  $-1$ , or  $x$  is  $-\text{Inf}$ .

41189 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41190 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41191 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41192 shall be raised.

41193 Pole Error The value of  $x$  is  $-1$ .

41194 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41195 then *errno* shall be set to [ERANGE]. If the integer expression  
 41196 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41197 floating-point exception shall be raised.

41198 These functions may fail if:

41199 MX Range Error The value of  $x$  is subnormal.

41200 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41201 then *errno* shall be set to [ERANGE]. If the integer expression  
 41202 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 41203 floating-point exception shall be raised.

**log1p()**41204 **EXAMPLES**

41205 None.

41206 **APPLICATION USAGE**41207 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41208 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41209 **RATIONALE**

41210 None.

41211 **FUTURE DIRECTIONS**

41212 None.

41213 **SEE ALSO**41214 *feclearexcept()*, *fetestexcept()*, *log()*

41215 XBD Section 4.19 (on page 116), &lt;math.h&gt;

41216 **CHANGE HISTORY**

41217 First released in Issue 4, Version 2.

41218 **Issue 5**

41219 Moved from X/OPEN UNIX extension to BASE.

41220 **Issue 6**

41221 The normative text is updated to avoid use of the term "must" for application requirements.

41222 The *log1p()* function is no longer marked as an extension.41223 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999  
41224 standard.41225 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
41226 revised to align with the ISO/IEC 9899:1999 standard.41227 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
41228 marked.

41229 **NAME**

41230 log2, log2f, log2l — compute base 2 logarithm functions

41231 **SYNOPSIS**

```
41232 #include <math.h>
41233 double log2(double x);
41234 float log2f(float x);
41235 long double log2l(long double x);
```

41236 **DESCRIPTION**

41237 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41238 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41239 volume of POSIX.1-2008 defers to the ISO C standard.

41240 These functions shall compute the base 2 logarithm of their argument  $x$ ,  $\log_2(x)$ .

41241 An application wishing to check for error situations should set *errno* to zero and call  
 41242 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41243 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41244 zero, an error has occurred.

41245 **RETURN VALUE**

41246 Upon successful completion, these functions shall return the base 2 logarithm of  $x$ .

41247 If  $x$  is  $\pm 0$ , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return  $-\text{HUGE\_VAL}$ ,  
 41248  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

41249 MX For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 41250 NaN (if supported), or an implementation-defined value shall be returned.

41251 MX If  $x$  is NaN, a NaN shall be returned.

41252 If  $x$  is 1, +0 shall be returned.

41253 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

41254 **ERRORS**

41255 These functions shall fail if:

41256 MX Domain Error The finite value of  $x$  is less than zero, or  $x$  is  $-\text{Inf}$ .

41257 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41258 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41259 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41260 shall be raised.

41261 Pole Error The value of  $x$  is zero.

41262 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41263 then *errno* shall be set to [ERANGE]. If the integer expression  
 41264 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41265 floating-point exception shall be raised.

**log2()**41266 **EXAMPLES**

41267 None.

41268 **APPLICATION USAGE**41269 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41270 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41271 **RATIONALE**

41272 None.

41273 **FUTURE DIRECTIONS**

41274 None.

41275 **SEE ALSO**41276 *feclearexcept()*, *fetestexcept()*, *log()*41277 XBD Section 4.19 (on page 116), <[math.h](#)>41278 **CHANGE HISTORY**

41279 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

41280 **NAME**

41281 logb, logbf, logbl — radix-independent exponent

41282 **SYNOPSIS**

```
41283 #include <math.h>
41284 double logb(double x);
41285 float logbf(float x);
41286 long double logbl(long double x);
```

41287 **DESCRIPTION**

41288 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41289 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41290 volume of POSIX.1-2008 defers to the ISO C standard.

41291 These functions shall compute the exponent of  $x$ , which is the integral part of  $\log_r |x|$ , as a  
 41292 signed floating-point value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating-point  
 41293 arithmetic, which is the value of FLT\_RADIX defined in the <float.h> header.

41294 If  $x$  is subnormal it is treated as though it were normalized; thus for finite positive  $x$ :

41295  $1 \leq x * \text{FLT\_RADIX}^{-\text{logb}(x)} < \text{FLT\_RADIX}$

41296 An application wishing to check for error situations should set *errno* to zero and call  
 41297 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41298 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41299 zero, an error has occurred.

41300 **RETURN VALUE**

41301 Upon successful completion, these functions shall return the exponent of  $x$ .

41302 If  $x$  is  $\pm 0$ , *logb()*, *logbf()*, and *logbl()* shall return `-HUGE_VAL`, `-HUGE_VALF`, and  
 41303 MX `-HUGE_VALL`, respectively. On systems that support the IEC 60559 Floating-Point option, a  
 41304 pole error shall occur;

41305 CX otherwise, a pole error may occur.

41306 MX If  $x$  is NaN, a NaN shall be returned.

41307 MX If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

41308 **ERRORS**

41309 These functions shall fail if:

41310 MX **Pole Error** The value of  $x$  is  $\pm 0$ .

41311 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41312 then *errno* shall be set to [ERANGE]. If the integer expression  
 41313 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41314 floating-point exception shall be raised.

41315 These functions may fail if:

41316 Pole Error The value of  $x$  is 0.

41317 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41318 then *errno* shall be set to [ERANGE]. If the integer expression  
 41319 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 41320 floating-point exception shall be raised.

**logb()**41321 **EXAMPLES**

41322           None.

41323 **APPLICATION USAGE**41324           On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41325           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41326 **RATIONALE**

41327           None.

41328 **FUTURE DIRECTIONS**

41329           None.

41330 **SEE ALSO**41331           *feclearexcept()*, *fetestexcept()*, *ilogb()*, *scalbln()*41332           XBD Section 4.19 (on page 116), [<float.h>](#), [<math.h>](#)41333 **CHANGE HISTORY**

41334           First released in Issue 4, Version 2.

41335 **Issue 5**

41336           Moved from X/OPEN UNIX extension to BASE.

41337 **Issue 6**41338           The *logb()* function is no longer marked as an extension.41339           The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.41340           The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
41341           revised to align with the ISO/IEC 9899:1999 standard.41342           IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
41343           marked.41344 **Issue 7**

41345           ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

*System Interfaces***logf()**41346 **NAME**41347 `logf, logl` — natural logarithm function41348 **SYNOPSIS**41349 `#include <math.h>`41350 `float logf(float x);`41351 `long double logl(long double x);`41352 **DESCRIPTION**41353 Refer to *log()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

# longjmp()

**41354 NAME**

41355           longjmp — non-local goto

**41356 SYNOPSIS**

41357           #include &lt;setjmp.h&gt;

41358           void longjmp(jmp\_buf env, int val);

**41359 DESCRIPTION**

41360 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 41361       conflict between the requirements described here and the ISO C standard is unintentional. This  
 41362       volume of POSIX.1-2008 defers to the ISO C standard.

41363       The *longjmp()* function shall restore the environment saved by the most recent invocation of  
 41364       *setjmp()* in the same thread, with the corresponding **jmp\_buf** argument. If there is no such  
 41365       invocation, or if the function containing the invocation of *setjmp()* has terminated execution in  
 41366       the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably  
 41367 CX       modified type and execution has left that scope in the interim, the behavior is undefined. It is  
 41368       unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or  
 41369       restores it to its value at the time *setjmp()* was called.

41370       All accessible objects have values, and all other components of the abstract machine have state  
 41371       (for example, floating-point status flags and open files), as of the time *longjmp()* was called,  
 41372       except that the values of objects of automatic storage duration are unspecified if they meet all  
 41373       the following conditions:

- 41374           • They are local to the function containing the corresponding *setjmp()* invocation.
- 41375           • They do not have volatile-qualified type.
- 41376           • They are changed between the *setjmp()* invocation and *longjmp()* call.

41377 CX       As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly  
 41378       in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is  
 41379       invoked from a nested signal handler (that is, from a function invoked as a result of a signal  
 41380       raised during the handling of another signal), the behavior is undefined.

41381       The effect of a call to *longjmp()* where initialization of the **jmp\_buf** structure was not performed  
 41382       in the calling thread is undefined.

**41383 RETURN VALUE**

41384       After *longjmp()* is completed, program execution continues as if the corresponding invocation of  
 41385       *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause  
 41386       *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

**41387 ERRORS**

41388       No errors are defined.

**41389 EXAMPLES**

41390       None.

**41391 APPLICATION USAGE**

41392       Applications whose behavior depends on the value of the signal mask should not use *longjmp()*  
 41393       and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the  
 41394       *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under  
 41395       application control).

41396 **RATIONALE**

41397 None.

41398 **FUTURE DIRECTIONS**

41399 None.

41400 **SEE ALSO**41401 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*

41402 XBD &lt;setjmp.h&gt;

41403 **CHANGE HISTORY**

41404 First released in Issue 1. Derived from Issue 1 of the SVID.

41405 **Issue 5**

41406 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

41407 **Issue 6**

41408 Extensions beyond the ISO C standard are marked.

41409 The following new requirements on POSIX implementations derive from alignment with the  
41410 Single UNIX Specification:

- 41411
- The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask  
41412 unspecified.

41413 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

**lrand48()**

System Interfaces

41414 **NAME**

41415 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

41416 **SYNOPSIS**

```
41417 XSI #include <stdlib.h>  
41418 long lrand48(void);
```

41419 **DESCRIPTION**41420 Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

41421 **NAME**

41422 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

41423 **SYNOPSIS**

```
41424 #include <math.h>
41425 long lrint(double x);
41426 long lrintf(float x);
41427 long lrintl(long double x);
```

41428 **DESCRIPTION**

41429 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41430 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41431 volume of POSIX.1-2008 defers to the ISO C standard.

41432 These functions shall round their argument to the nearest integer value, rounding according to  
 41433 the current rounding direction.

41434 An application wishing to check for error situations should set *errno* to zero and call  
 41435 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41436 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41437 zero, an error has occurred.

41438 **RETURN VALUE**

41439 Upon successful completion, these functions shall return the rounded integer value.

41440 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.41441 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.41442 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

41443 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be  
 41444 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall  
 41445 occur;

41446 CX otherwise, a domain error may occur.

41447 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be  
 41448 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall  
 41449 occur;

41450 CX otherwise, a domain error may occur.

41451 **ERRORS**

41452 These functions shall fail if:

41453 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 41454 integer.

41455 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41456 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41457 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41458 shall be raised.

41459 These functions may fail if:

41460 **Domain Error** The correct value is not representable as an integer.

41461 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41462 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41463 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41464 shall be raised.

**lrint()**41465 **EXAMPLES**

41466 None.

41467 **APPLICATION USAGE**41468 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41469 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41470 **RATIONALE**41471 These functions provide floating-to-integer conversions. They round according to the current  
41472 rounding direction. If the rounded value is outside the range of the return type, the numeric  
41473 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
41474 floating-point exception and the result differs from the argument, they raise the inexact floating-  
41475 point exception.41476 **FUTURE DIRECTIONS**

41477 None.

41478 **SEE ALSO**41479 *feclearexcept()*, *fetestexcept()*, *llrint()*41480 XBD Section 4.19 (on page 116), [<math.h>](#)41481 **CHANGE HISTORY**

41482 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

41483 **Issue 7**

41484 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

41485 **NAME**

41486 lround, lroundf, lroundl — round to nearest integer value

41487 **SYNOPSIS**

```
41488 #include <math.h>
41489 long lround(double x);
41490 long lroundf(float x);
41491 long lroundl(long double x);
```

41492 **DESCRIPTION**

41493 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 41494 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41495 volume of POSIX.1-2008 defers to the ISO C standard.

41496 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 41497 away from zero, regardless of the current rounding direction.

41498 An application wishing to check for error situations should set *errno* to zero and call  
 41499 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41500 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41501 zero, an error has occurred.

41502 **RETURN VALUE**

41503 Upon successful completion, these functions shall return the rounded integer value.

41504 **MX** If *x* is NaN, a domain error shall occur and an unspecified value is returned.

41505 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

41506 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

41507 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be  
 41508 **MX** returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;  
 41509 **CX** otherwise, a domain error may occur.

41510 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be  
 41511 **MX** returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;  
 41512 **CX** otherwise, a domain error may occur.

41513 **ERRORS**

41514 These functions shall fail if:

41515 **MX** **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 41516 integer.

41517 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41518 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41519 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41520 shall be raised.

41521 These functions may fail if:

41522 **Domain Error** The correct value is not representable as an integer.

41523 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41524 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 41525 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 41526 shall be raised.

**lround()**41527 **EXAMPLES**

41528       None.

41529 **APPLICATION USAGE**41530       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
41531       MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41532 **RATIONALE**41533       These functions differ from the *lrint()* functions in the default rounding direction, with the  
41534       *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact  
41535       floating-point exception for non-integer arguments that round to within the range of the return  
41536       type.41537 **FUTURE DIRECTIONS**

41538       None.

41539 **SEE ALSO**41540       *feclearexcept()*, *fetestexcept()*, *llround()*

41541       XBD Section 4.19 (on page 116), &lt;math.h&gt;

41542 **CHANGE HISTORY**

41543       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

41544 **Issue 7**

41545       ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

41546 **NAME**

41547 lsearch, lfind — linear search and update

41548 **SYNOPSIS**

```

41549 XSI #include <search.h>
41550 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
41551             int (*compar)(const void *, const void *));
41552 void *lfind(const void *key, const void *base, size_t *nel,
41553            size_t width, int (*compar)(const void *, const void *));

```

41554 **DESCRIPTION**

41555 The *lsearch()* function shall linearly search the table and return a pointer into the table for the  
 41556 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*  
 41557 argument points to the entry to be sought in the table. The *base* argument points to the first  
 41558 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument  
 41559 points to an integer containing the current number of elements in the table. The integer to which  
 41560 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to  
 41561 a comparison function which the application shall supply (for example, *strcmp()*). It is called  
 41562 with two arguments that point to the elements being compared. The application shall ensure  
 41563 that the function returns 0 if the elements are equal, and non-zero otherwise.

41564 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not  
 41565 added to the table. Instead, a null pointer is returned.

41566 **RETURN VALUE**

41567 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it.  
 41568 Otherwise, *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly  
 41569 added element.

41570 Both functions shall return a null pointer in case of error.

41571 **ERRORS**

41572 No errors are defined.

41573 **EXAMPLES**41574 **Storing Strings in a Table**

41575 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to  
 41576 ELSIZE and stores them in a table, eliminating duplicates.

```

41577 #include <stdio.h>
41578 #include <string.h>
41579 #include <search.h>
41580 #define TABSIZE 50
41581 #define ELSIZE 120
41582 ...
41583     char line[ELSIZE], tab[TABSIZE][ELSIZE];
41584     size_t nel = 0;
41585     ...
41586     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
41587         (void) lsearch(line, tab, &nel,
41588                      ELSIZE, (int (*)(const void *, const void *)) strcmp);
41589     ...

```

**lsearch()**41590 **Finding a Matching Entry**

41591 The following example finds any line that reads "This is a test."

```
41592 #include <search.h>
41593 #include <string.h>
41594 ...
41595 char line[ELSIZE], tab[TABSIZE][ELSIZE];
41596 size_t nel = 0;
41597 char *findline;
41598 void *entry;

41599 findline = "This is a test.\n";

41600 entry = lfind(findline, tab, &nel, ELSIZE, (
41601     int (*)(const void *, const void *)) strcmp);
```

41602 **APPLICATION USAGE**

41603 The comparison function need not compare every byte, so arbitrary data may be contained in  
41604 the elements in addition to the values being compared.

41605 Undefined results can occur if there is not enough room in the table to add a new item.

41606 **RATIONALE**

41607 None.

41608 **FUTURE DIRECTIONS**

41609 None.

41610 **SEE ALSO**

41611 *hcreate()*, *tdelete()*

41612 XBD [<search.h>](#)

41613 **CHANGE HISTORY**

41614 First released in Issue 1. Derived from Issue 1 of the SVID.

41615 **Issue 6**

41616 The normative text is updated to avoid use of the term "must" for application requirements.

41617 **NAME**

41618 lseek — move the read/write file offset

41619 **SYNOPSIS**

41620 #include &lt;unistd.h&gt;

41621 off\_t lseek(int *fildes*, off\_t *offset*, int *whence*);41622 **DESCRIPTION**41623 The *lseek()* function shall set the file offset for the open file description associated with the file  
41624 descriptor *fildes*, as follows:

- 41625 • If *whence* is SEEK\_SET, the file offset shall be set to *offset* bytes.
- 41626 • If *whence* is SEEK\_CUR, the file offset shall be set to its current location plus *offset*.
- 41627 • If *whence* is SEEK\_END, the file offset shall be set to the size of the file plus *offset*.

41628 The symbolic constants SEEK\_SET, SEEK\_CUR, and SEEK\_END are defined in &lt;unistd.h&gt;.

41629 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.  
41630 The value of the file offset associated with such a device is undefined.41631 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the  
41632 file. If data is later written at this point, subsequent reads of data in the gap shall return bytes  
41633 with the value 0 until data is actually written into the gap.41634 The *lseek()* function shall not, by itself, extend the size of a file.41635 SHM If *fildes* refers to a shared memory object, the result of the *lseek()* function is unspecified.41636 TYP If *fildes* refers to a typed memory object, the result of the *lseek()* function is unspecified.41637 **RETURN VALUE**41638 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the  
41639 file, shall be returned. Otherwise, (off\_t)-1 shall be returned, *errno* shall be set to indicate the  
41640 error, and the file offset shall remain unchanged.41641 **ERRORS**41642 The *lseek()* function shall fail if:

- |       |             |  |
|-------|-------------|--|
| 41643 | [EBADF]     | The <i>fildes</i> argument is not an open file descriptor.   |
| 41644 | [EINVAL]    | The <i>whence</i> argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory. |
| 41646 | [EOVERFLOW] | The resulting file offset would be a value which cannot be represented correctly in an object of type <b>off_t</b> .                                   |
| 41648 | [ESPIPE]    | The <i>fildes</i> argument is associated with a pipe, FIFO, or socket.   |

41649 **EXAMPLES**

41650 None.

41651 **APPLICATION USAGE**

41652 None.

41653 **RATIONALE**41654 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files  
41655 by use of a special positioning type.41656 Although *lseek()* may position the file offset beyond the end of the file, this function does not  
41657 itself extend the size of the file. While the only function in POSIX.1-2008 that may directly extend

**lseek()**

41658 the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally derived from  
 41659 the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing calls on *write()*).

41660 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-  
 41661 defined and device-dependent (for example, memory may have few invalid values). A negative  
 41662 file offset may be valid for some devices in some implementations.

41663 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.  
 41664 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return  
 41665 to determine whether a return value of (**off\_t**)-1 is a negative offset or an indication of an error  
 41666 condition. The standard developers did not wish to require this action on the part of a  
 41667 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting  
 41668 file offset would be negative for a regular file, block special file, or directory.

**FUTURE DIRECTIONS**

41669 None.

**SEE ALSO**

41672 *open()*

41673 XBD <**sys/types.h**>, <**unistd.h**>

**CHANGE HISTORY**

41674 First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

41677 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

41678 Large File Summit extensions are added.

**Issue 6**

41680 In the SYNOPSIS, the optional include of the <**sys/types.h**> header is removed.

41681 The following new requirements on POSIX implementations derive from alignment with the  
 41682 Single UNIX Specification:

- 41683 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
 41684 required for conforming implementations of previous POSIX specifications, it was not  
 41685 required for UNIX applications.
- 41686 • The [EOVERFLOW] error condition is added. This change is to support large files.

41687 An additional [ESPIPE] error condition is added for sockets.

41688 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 41689 *lseek()* results are unspecified for typed memory objects.

*System Interfaces***lstat()**41690 **NAME**

41691        lstat — get file status

41692 **SYNOPSIS**

41693        #include &lt;sys/stat.h&gt;

41694        int lstat(const char \*restrict *path*, struct stat \*restrict *buf*);41695 **DESCRIPTION**41696        Refer to *fstatat()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**malloc()**41697 **NAME**41698 `malloc` — a memory allocator41699 **SYNOPSIS**41700 `#include <stdlib.h>`41701 `void *malloc(size_t size);`41702 **DESCRIPTION**

41703 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41704 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41705 volume of POSIX.1-2008 defers to the ISO C standard.

41706 The `malloc()` function shall allocate unused space for an object whose size in bytes is specified by  
 41707 `size` and whose value is unspecified.

41708 The order and contiguity of storage allocated by successive calls to `malloc()` is unspecified. The  
 41709 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 41710 a pointer to any type of object and then used to access such an object in the space allocated (until  
 41711 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
 41712 disjoint from any other object. The pointer returned points to the start (lowest byte address) of  
 41713 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of  
 41714 the space requested is 0, the behavior is implementation-defined: the value returned shall be  
 41715 either a null pointer or a unique pointer.

41716 **RETURN VALUE**

41717 Upon successful completion with `size` not equal to 0, `malloc()` shall return a pointer to the  
 41718 allocated space. If `size` is 0, either a null pointer or a unique pointer that can be successfully  
 41719 CX passed to `free()` shall be returned. Otherwise, it shall return a null pointer and set `errno` to  
 41720 indicate the error.

41721 **ERRORS**41722 The `malloc()` function shall fail if:

41723 CX [ENOMEM] Insufficient storage space is available.

41724 **EXAMPLES**

41725 None.

41726 **APPLICATION USAGE**

41727 None.

41728 **RATIONALE**

41729 None.

41730 **FUTURE DIRECTIONS**

41731 None.

41732 **SEE ALSO**41733 `calloc()`, `free()`, `getrlimit()`, `posix_memalign()`, `realloc()`41734 XBD `<stdlib.h>`41735 **CHANGE HISTORY**

41736 First released in Issue 1. Derived from Issue 1 of the SVID.

41737 **Issue 6**

41738 Extensions beyond the ISO C standard are marked.

- 41739 The following new requirements on POSIX implementations derive from alignment with the  
41740 Single UNIX Specification:
- 41741 • In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
  - 41742 • The [ENOMEM] error condition is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**mblen()**41743 **NAME**41744 `mblen` — get number of bytes in a character41745 **SYNOPSIS**41746 `#include <stdlib.h>`41747 `int mblen(const char *s, size_t n);`41748 **DESCRIPTION**41749 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
41750 conflict between the requirements described here and the ISO C standard is unintentional. This  
41751 volume of POSIX.1-2008 defers to the ISO C standard.41752 If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character  
41753 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:41754 `mbtowc((wchar_t *)0, s, n);`41755 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
41756 *mblen()*.41757 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
41758 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
41759 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
41760 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
41761 null pointer shall cause this function to return a non-zero value if encodings have state  
41762 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
41763 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
41764 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
41765 unspecified.41766 **RETURN VALUE**41767 If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,  
41768 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall  
41769 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
41770 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a  
41771 valid character) and may set *errno* to indicate the error. In no case shall the value returned be  
41772 greater than *n* or the value of the {MB\_CUR\_MAX} macro.41773 **ERRORS**41774 The *mblen()* function may fail if:

41775 XSI [EILSEQ] An invalid character sequence is detected.

41776 **EXAMPLES**

41777 None.

41778 **APPLICATION USAGE**

41779 None.

41780 **RATIONALE**

41781 None.

41782 **FUTURE DIRECTIONS**

41783 None.

41784 **SEE ALSO**41785 [mbtowc\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)41786 XBD [<stdlib.h>](#)41787 **CHANGE HISTORY**

41788 First released in Issue 4. Aligned with the ISO C standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**mbrlen()**41789 **NAME**41790 `mbrlen` — get number of bytes in a character (restartable)41791 **SYNOPSIS**41792 `#include <wchar.h>`41793 `size_t mbrlen(const char *restrict s, size_t n,`  
41794 `mbstate_t *restrict ps);`41795 **DESCRIPTION**41796 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
41797 conflict between the requirements described here and the ISO C standard is unintentional. This  
41798 volume of POSIX.1-2008 defers to the ISO C standard.41799 If *s* is not a null pointer, *mbrlen()* shall determine the number of bytes constituting the character  
41800 pointed to by *s*. It shall be equivalent to:41801 `mbstate_t internal;`  
41802 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`41803 If *ps* is a null pointer, the *mbrlen()* function shall use its own internal **mbstate\_t** object, which is  
41804 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
41805 pointed to by *ps* shall be used to completely describe the current conversion state of the  
41806 associated character sequence. The implementation shall behave as if no function defined in this  
41807 volume of POSIX.1-2008 calls *mbrlen()*.41808 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.41809 **RETURN VALUE**41810 The *mbrlen()* function shall return the first of the following that applies:41811 `0` If the next *n* or fewer bytes complete the character that corresponds to the null  
41812 wide character.41813 *positive* If the next *n* or fewer bytes complete a valid character; the value returned shall  
41814 be the number of bytes that complete the character.41815 `(size_t)-2` If the next *n* bytes contribute to an incomplete but potentially valid character,  
41816 and all *n* bytes have been processed. When *n* has at least the value of the  
41817 `{MB_CUR_MAX}` macro, this case can only occur if *s* points at a sequence of  
41818 redundant shift sequences (for implementations with state-dependent  
41819 encodings).41820 `(size_t)-1` If an encoding error occurs, in which case the next *n* or fewer bytes do not  
41821 contribute to a complete and valid character. In this case, `[EILSEQ]` shall be  
41822 stored in *errno* and the conversion state is undefined.41823 **ERRORS**41824 The *mbrlen()* function shall fail if:41825 `[EILSEQ]` An invalid character sequence is detected.41826 The *mbrlen()* function may fail if:41827 `[EINVAL]` *ps* points to an object that contains an invalid conversion state.

41828 **EXAMPLES**

41829 None.

41830 **APPLICATION USAGE**

41831 None.

41832 **RATIONALE**

41833 None.

41834 **FUTURE DIRECTIONS**

41835 None.

41836 **SEE ALSO**41837 [mbsinit\(\)](#), [mbrtowc\(\)](#)41838 XBD [<wchar.h>](#)41839 **CHANGE HISTORY**41840 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
41841 (E).41842 **Issue 6**41843 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.41844 **Issue 7**

41845 Austin Group Interpretation 1003.1-2001 #170 is applied.

**mbrtowc()**41846 **NAME**41847 `mbrtowc` — convert a character to a wide-character code (restartable)41848 **SYNOPSIS**41849 `#include <wchar.h>`41850 `size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,`  
41851 `size_t n, mbstate_t *restrict ps);`41852 **DESCRIPTION**41853 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
41854 conflict between the requirements described here and the ISO C standard is unintentional. This  
41855 volume of POSIX.1-2008 defers to the ISO C standard.41856 If *s* is a null pointer, the `mbrtowc()` function shall be equivalent to the call:41857 `mbrtowc(NULL, "", 1, ps)`41858 In this case, the values of the arguments *pwc* and *n* are ignored.41859 If *s* is not a null pointer, the `mbrtowc()` function shall inspect at most *n* bytes beginning at the  
41860 byte pointed to by *s* to determine the number of bytes needed to complete the next character  
41861 (including any shift sequences). If the function determines that the next character is completed,  
41862 it shall determine the value of the corresponding wide character and then, if *pwc* is not a null  
41863 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide  
41864 character is the null wide character, the resulting state described shall be the initial conversion  
41865 state.41866 If *ps* is a null pointer, the `mbrtowc()` function shall use its own internal **mbstate\_t** object, which  
41867 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t**  
41868 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
41869 associated character sequence. The implementation shall behave as if no function defined in this  
41870 volume of POSIX.1-2008 calls `mbrtowc()`.41871 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.41872 **RETURN VALUE**41873 The `mbrtowc()` function shall return the first of the following that applies:41874 **0** If the next *n* or fewer bytes complete the character that corresponds to the null  
41875 wide character (which is the value stored).41876 between 1 and *n* inclusive41877 If the next *n* or fewer bytes complete a valid character (which is the value  
41878 stored); the value returned shall be the number of bytes that complete the  
41879 character.41880 **(size\_t)-2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
41881 and all *n* bytes have been processed (no value is stored). When *n* has at least  
41882 the value of the `{MB_CUR_MAX}` macro, this case can only occur if *s* points at  
41883 a sequence of redundant shift sequences (for implementations with state-  
41884 dependent encodings).41885 **(size\_t)-1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
41886 contribute to a complete and valid character (no value is stored). In this case,  
41887 `[EILSEQ]` shall be stored in `errno` and the conversion state is undefined.

41888 **ERRORS**41889 The *mbrtowc()* function shall fail if:

41890 [EILSEQ] An invalid character sequence is detected.

41891 The *mbrtowc()* function may fail if:41892 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.41893 **EXAMPLES**

41894 None.

41895 **APPLICATION USAGE**

41896 None.

41897 **RATIONALE**

41898 None.

41899 **FUTURE DIRECTIONS**

41900 None.

41901 **SEE ALSO**41902 *mbsinit()*, *mbsrtowcs()*41903 XBD <*wchar.h*>41904 **CHANGE HISTORY**41905 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
41906 (E).41907 **Issue 6**41908 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.41909 The following new requirements on POSIX implementations derive from alignment with the  
41910 Single UNIX Specification:

- 41911
- The [EINVAL] error condition is added.

41912 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

41913 **Issue 7**

41914 Austin Group Interpretation 1003.1-2001 #170 is applied.

**mbsinit()**41915 **NAME**41916 `mbsinit` — determine conversion object status41917 **SYNOPSIS**

```
41918 #include <wchar.h>
41919 int mbsinit(const mbstate_t *ps);
```

41920 **DESCRIPTION**

41921 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41922 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41923 volume of POSIX.1-2008 defers to the ISO C standard.

41924 If *ps* is not a null pointer, the `mbsinit()` function shall determine whether the object pointed to by  
 41925 *ps* describes an initial conversion state.

41926 **RETURN VALUE**

41927 The `mbsinit()` function shall return non-zero if *ps* is a null pointer, or if the pointed-to object  
 41928 describes an initial conversion state; otherwise, it shall return zero.

41929 If an `mbstate_t` object is altered by any of the functions described as “restartable”, and is then  
 41930 used with a different character sequence, or in the other conversion direction, or with a different  
 41931 `LC_CTYPE` category setting than on earlier function calls, the behavior is undefined.

41932 **ERRORS**

41933 No errors are defined.

41934 **EXAMPLES**

41935 None.

41936 **APPLICATION USAGE**

41937 The `mbstate_t` object is used to describe the current conversion state from a particular character  
 41938 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the  
 41939 `LC_CTYPE` category of the current locale.

41940 The initial conversion state corresponds, for a conversion in either direction, to the beginning of  
 41941 a new character sequence in the initial shift state. A zero valued `mbstate_t` object is at least one  
 41942 way to describe an initial conversion state. A zero valued `mbstate_t` object can be used to initiate  
 41943 conversion involving any character sequence, in any `LC_CTYPE` category setting.

41944 **RATIONALE**

41945 None.

41946 **FUTURE DIRECTIONS**

41947 None.

41948 **SEE ALSO**41949 `mbilen()`, `mbrtowc()`, `mbsrtowcs()`, `wcrtomb()`, `wcsrtombs()`41950 XBD `<wchar.h>`41951 **CHANGE HISTORY**

41952 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 41953 (E).

41954 **NAME**41955 `mbsnrto wcs, mbsrtowcs` — convert a character string to a wide-character string (restartable)41956 **SYNOPSIS**41957 `#include <wchar.h>`

```
41958 CX size_t mbsnrto wcs(wchar_t *restrict dst, const char **restrict src,
41959 size_t nmc, size_t len, mbstate_t *restrict ps);
41960 size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
41961 size_t len, mbstate_t *restrict ps);
```

41962 **DESCRIPTION**

41963 CX For `mbsrtowcs()`: The functionality described on this reference page is aligned with the ISO C  
 41964 standard. Any conflict between the requirements described here and the ISO C standard is  
 41965 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

41966 The `mbsrtowcs()` function shall convert a sequence of characters, beginning in the conversion  
 41967 state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a  
 41968 sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters  
 41969 shall be stored into the array pointed to by `dst`. Conversion continues up to and including a  
 41970 terminating null character, which shall also be stored. Conversion shall stop early in either of the  
 41971 following cases:

- 41972 • A sequence of bytes is encountered that does not form a valid character.
- 41973 • `len` codes have been stored into the array pointed to by `dst` (and `dst` is not a null pointer).

41974 Each conversion shall take place as if by a call to the `mbrtowc()` function.

41975 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null  
 41976 pointer (if conversion stopped due to reaching a terminating null character) or the address just  
 41977 past the last character converted (if any). If conversion stopped due to reaching a terminating  
 41978 null character, and if `dst` is not a null pointer, the resulting state described shall be the initial  
 41979 conversion state.

41980 If `ps` is a null pointer, the `mbsrtowcs()` function shall use its own internal `mbstate_t` object, which  
 41981 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
 41982 pointed to by `ps` shall be used to completely describe the current conversion state of the  
 41983 associated character sequence.

41984 CX The `mbsnrto wcs()` function shall be equivalent to the `mbsrtowcs()` function, except that the  
 41985 conversion of characters pointed to by `src` is limited to at most `nmc` bytes (the size of the input  
 41986 buffer).

41987 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

41988 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
 41989 these functions.

41990 **RETURN VALUE**

41991 If the input conversion encounters a sequence of bytes that do not form a valid character, an  
 41992 encoding error occurs. In this case, these functions shall store the value of the macro `[EILSEQ]` in  
 41993 `errno` and shall return `(size_t)-1`; the conversion state is undefined. Otherwise, these functions  
 41994 shall return the number of characters successfully converted, not including the terminating null  
 41995 (if any).

**mbsrtowcs()**41996 **ERRORS**

41997 These functions shall fail if:

41998 [EILSEQ] An invalid character sequence is detected.

41999 These functions may fail if:

42000 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.42001 **EXAMPLES**

42002 None.

42003 **APPLICATION USAGE**

42004 None.

42005 **RATIONALE**

42006 None.

42007 **FUTURE DIRECTIONS**

42008 None.

42009 **SEE ALSO**42010 *iconv()*, *mbrtowc()*, *mbstowcs()*42011 XBD <*wchar.h*>42012 **CHANGE HISTORY**42013 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
42014 (E).42015 **Issue 6**42016 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42017 The [EINVAL] error condition is marked CX.

42018 **Issue 7**

42019 Austin Group Interpretation 1003.1-2001 #170 is applied.

42020 The *mbsnr towcs()* function is added from The Open Group Technical Standard, 2006, Extended  
42021 API Set Part 1.

42022 **NAME**

42023 mbstowcs — convert a character string to a wide-character string

42024 **SYNOPSIS**

42025 #include &lt;stdlib.h&gt;

42026 size\_t mbstowcs(wchar\_t \*restrict pwcs, const char \*restrict s,  
42027 size\_t n);42028 **DESCRIPTION**42029 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
42030 conflict between the requirements described here and the ISO C standard is unintentional. This  
42031 volume of POSIX.1-2008 defers to the ISO C standard.42032 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift  
42033 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and  
42034 shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No  
42035 characters that follow a null byte (which is converted into a wide-character code with value 0)  
42036 shall be examined or converted. Each character shall be converted as if by a call to *mbtowlc()*,  
42037 except that the shift state of *mbtowlc()* is not affected.42038 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes  
42039 place between objects that overlap, the behavior is undefined.42040 XSI The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.  
42041 If *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array  
42042 regardless of the value of *n*, but no values are stored.42043 **RETURN VALUE**42044 CX If an invalid character is encountered, *mbstowcs()* shall return **(size\_t)-1** and may set *errno* to  
42045 indicate the error.42046 XSI Otherwise, *mbstowcs()* shall return the number of the array elements modified (or required if  
42047 *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if  
42048 the value returned is *n*.42049 **ERRORS**42050 The *mbstowcs()* function shall fail if:

42051 XSI [EILSEQ] An invalid byte sequence is detected.

42052 **EXAMPLES**

42053 None.

42054 **APPLICATION USAGE**

42055 None.

42056 **RATIONALE**

42057 None.

42058 **FUTURE DIRECTIONS**

42059 None.

42060 **SEE ALSO**42061 [mblen\(\)](#), [mbtowlc\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)42062 XBD [<stdlib.h>](#)

**mbstowcs()***System Interfaces*42063 **CHANGE HISTORY**

42064 First released in Issue 4. Aligned with the ISO C standard.

42065 **Issue 6**42066 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42067 Extensions beyond the ISO C standard are marked.

42068 **Issue 7**

42069 Austin Group Interpretation 1003.1-2001 #170 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42070 **NAME**

42071 mbtowc — convert a character to a wide-character code

42072 **SYNOPSIS**

42073 #include &lt;stdlib.h&gt;

42074 int mbtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*, size\_t *n*);42075 **DESCRIPTION**

42076 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42077 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42078 volume of POSIX.1-2008 defers to the ISO C standard.

42079 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the  
 42080 character pointed to by *s*. It shall then determine the wide-character code for the value of type  
 42081 **wchar\_t** that corresponds to that character. (The value of the wide-character code corresponding  
 42082 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store  
 42083 the wide-character code in the object pointed to by *pwc*.

42084 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 42085 state-dependent encoding, this function is placed into its initial state by a call for which its  
 42086 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 42087 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 42088 null pointer shall cause this function to return a non-zero value if encodings have state  
 42089 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
 42090 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
 42091 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
 42092 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

42093 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
 42094 *mbtowc()*.

42095 **RETURN VALUE**

42096 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,  
 42097 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*  
 42098 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 42099 converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may  
 42100 set *errno* to indicate the error (if they do not form a valid character).

42101 In no case shall the value returned be greater than *n* or the value of the {MB\_CUR\_MAX} macro.

42102 **ERRORS**

42103 The *mbtowc()* function shall fail if:

42104 XSI [EILSEQ] An invalid character sequence is detected.

42105 **EXAMPLES**

42106 None.

42107 **APPLICATION USAGE**

42108 None.

42109 **RATIONALE**

42110 None.

42111 **FUTURE DIRECTIONS**

42112 None.

**mbtowc()**

System Interfaces

42113 **SEE ALSO**42114 [mblen\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)42115 XBD [<stdlib.h>](#)42116 **CHANGE HISTORY**

42117 First released in Issue 4. Aligned with the ISO C standard.

42118 **Issue 6**42119 The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42120 Extensions beyond the ISO C standard are marked.

42121 **Issue 7**

42122 Austin Group Interpretation 1003.1-2001 #170 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42123 **NAME**

42124           memcpy — copy bytes in memory

42125 **SYNOPSIS**

```
42126 XSI      #include <string.h>
42127
42127 void *memcpy(void *restrict s1, const void *restrict s2,
42128             int c, size_t n);
```

42129 **DESCRIPTION**

42130       The *memcpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first  
42131       occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,  
42132       whichever comes first. If copying takes place between objects that overlap, the behavior is  
42133       undefined.

42134 **RETURN VALUE**

42135       The *memcpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null  
42136       pointer if *c* was not found in the first *n* bytes of *s2*.

42137 **ERRORS**

42138       No errors are defined.

42139 **EXAMPLES**

42140       None.

42141 **APPLICATION USAGE**

42142       The *memcpy()* function does not check for the overflow of the receiving memory area.

42143 **RATIONALE**

42144       None.

42145 **FUTURE DIRECTIONS**

42146       None.

42147 **SEE ALSO**

42148       XBD [<string.h>](#)

42149 **CHANGE HISTORY**

42150       First released in Issue 1. Derived from Issue 1 of the SVID.

42151 **Issue 6**

42152       The **restrict** keyword is added to the *memcpy()* prototype for alignment with the  
42153       ISO/IEC 9899:1999 standard.

**memchr()**42154 **NAME**

42155 memchr — find byte in memory

42156 **SYNOPSIS**

42157 #include &lt;string.h&gt;

42158 void \*memchr(const void \*s, int c, size\_t n);

42159 **DESCRIPTION**

42160 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42161 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42162 volume of POSIX.1-2008 defers to the ISO C standard.

42163 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in  
 42164 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

42165 **RETURN VALUE**

42166 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does  
 42167 not occur in the object.

42168 **ERRORS**

42169 No errors are defined.

42170 **EXAMPLES**

42171 None.

42172 **APPLICATION USAGE**

42173 None.

42174 **RATIONALE**

42175 None.

42176 **FUTURE DIRECTIONS**

42177 None.

42178 **SEE ALSO**

42179 XBD &lt;string.h&gt;

42180 **CHANGE HISTORY**

42181 First released in Issue 1. Derived from Issue 1 of the SVID.

42182 **NAME**

42183       memcmp — compare bytes in memory

42184 **SYNOPSIS**

42185       #include &lt;string.h&gt;

42186       int memcmp(const void \*s1, const void \*s2, size\_t n);

42187 **DESCRIPTION**42188 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
42189 conflict between the requirements described here and the ISO C standard is unintentional. This  
42190 volume of POSIX.1-2008 defers to the ISO C standard.42191       The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the  
42192 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.42193       The sign of a non-zero return value shall be determined by the sign of the difference between the  
42194 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects  
42195 being compared.42196 **RETURN VALUE**42197       The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object  
42198 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.42199 **ERRORS**

42200       No errors are defined.

42201 **EXAMPLES**

42202       None.

42203 **APPLICATION USAGE**

42204       None.

42205 **RATIONALE**

42206       None.

42207 **FUTURE DIRECTIONS**

42208       None.

42209 **SEE ALSO**

42210       XBD &lt;string.h&gt;

42211 **CHANGE HISTORY**

42212       First released in Issue 1. Derived from Issue 1 of the SVID.

**memcpy()**42213 **NAME**42214 `memcpy` — copy bytes in memory42215 **SYNOPSIS**42216 `#include <string.h>`42217 `void *memcpy(void *restrict s1, const void *restrict s2, size_t n);`42218 **DESCRIPTION**42219 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
42220 conflict between the requirements described here and the ISO C standard is unintentional. This  
42221 volume of POSIX.1-2008 defers to the ISO C standard.42222 The `memcpy()` function shall copy *n* bytes from the object pointed to by *s2* into the object pointed  
42223 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.42224 **RETURN VALUE**42225 The `memcpy()` function shall return *s1*; no return value is reserved to indicate an error.42226 **ERRORS**

42227 No errors are defined.

42228 **EXAMPLES**

42229 None.

42230 **APPLICATION USAGE**42231 The `memcpy()` function does not check for the overflow of the receiving memory area.42232 **RATIONALE**

42233 None.

42234 **FUTURE DIRECTIONS**

42235 None.

42236 **SEE ALSO**42237 XBD [<string.h>](#)42238 **CHANGE HISTORY**

42239 First released in Issue 1. Derived from Issue 1 of the SVID.

42240 **Issue 6**42241 The `memcpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42242 **NAME**

42243 memmove — copy bytes in memory with overlapping areas

42244 **SYNOPSIS**

42245 #include &lt;string.h&gt;

42246 void \*memmove(void \*s1, const void \*s2, size\_t n);

42247 **DESCRIPTION**

42248 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42249 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42250 volume of POSIX.1-2008 defers to the ISO C standard.

42251 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object  
 42252 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first  
 42253 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and  
 42254 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

42255 **RETURN VALUE**42256 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.42257 **ERRORS**

42258 No errors are defined.

42259 **EXAMPLES**

42260 None.

42261 **APPLICATION USAGE**

42262 None.

42263 **RATIONALE**

42264 None.

42265 **FUTURE DIRECTIONS**

42266 None.

42267 **SEE ALSO**

42268 XBD &lt;string.h&gt;

42269 **CHANGE HISTORY**

42270 First released in Issue 4. Derived from the ANSI C standard.

**memset()**42271 **NAME**42272           **memset** — set bytes in memory42273 **SYNOPSIS**

42274           #include &lt;string.h&gt;

42275           void \*memset(void \*s, int c, size\_t n);

42276 **DESCRIPTION**42277 **CX**           The functionality described on this reference page is aligned with the ISO C standard. Any  
42278 conflict between the requirements described here and the ISO C standard is unintentional. This  
42279 volume of POSIX.1-2008 defers to the ISO C standard.42280           The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes  
42281 of the object pointed to by *s*.42282 **RETURN VALUE**42283           The *memset()* function shall return *s*; no return value is reserved to indicate an error.42284 **ERRORS**

42285           No errors are defined.

42286 **EXAMPLES**

42287           None.

42288 **APPLICATION USAGE**

42289           None.

42290 **RATIONALE**

42291           None.

42292 **FUTURE DIRECTIONS**

42293           None.

42294 **SEE ALSO**

42295           XBD &lt;string.h&gt;

42296 **CHANGE HISTORY**

42297           First released in Issue 1. Derived from Issue 1 of the SVID.

42298 **NAME**

42299        mkdir, mkdirat — make a directory relative to directory file descriptor

42300 **SYNOPSIS**

42301        #include &lt;sys/stat.h&gt;

42302        int mkdir(const char \*path, mode\_t mode);

42303        int mkdirat(int fd, const char \*path, mode\_t mode);

42304 **DESCRIPTION**

42305        The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the  
 42306        new directory shall be initialized from *mode*. These file permission bits of the *mode* argument  
 42307        shall be modified by the process' file creation mask.

42308        When bits in *mode* other than the file permission bits are set, the meaning of these additional bits  
 42309        is implementation-defined.

42310        The directory's user ID shall be set to the process' effective user ID. The directory's group ID  
 42311        shall be set to the group ID of the parent directory or to the effective group ID of the process.  
 42312        Implementations shall provide a way to initialize the directory's group ID to the group ID of the  
 42313        parent directory. Implementations may, but need not, provide an implementation-defined way  
 42314        to initialize the directory's group ID to the effective group ID of the calling process.

42315        The newly created directory shall be an empty directory.

42316        If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

42317        Upon successful completion, *mkdir()* shall mark for update the last data access, last data  
 42318        modification, and last file status change timestamps of the directory. Also, the last data  
 42319        modification and last file status change timestamps of the directory that contains the new entry  
 42320        shall be marked for update.

42321        The *mkdirat()* function shall be equivalent to the *mkdir()* function except in the case where *path*  
 42322        specifies a relative path. In this case the newly created directory is created relative to the  
 42323        directory associated with the file descriptor *fd* instead of the current working directory. If the file  
 42324        descriptor was opened without O\_SEARCH, the function shall check whether directory searches  
 42325        are permitted using the current permissions of the directory underlying the file descriptor. If the  
 42326        file descriptor was opened with O\_SEARCH, the function shall not perform the check.

42327        If *mkdirat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 42328        directory is used and the behavior shall be identical to a call to *mkdir()*.

42329 **RETURN VALUE**

42330        Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 42331        return -1 and set *errno* to indicate the error. If -1 is returned, no directory shall be created.

42332 **ERRORS**

42333        These functions shall fail if:

- |       |          |  |
|-------|----------|--|
| 42334 | [EACCES] | Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created. |
| 42335 |          |  |
| 42336 | [EEXIST] | The named file exists.   |
| 42337 | [ELOOP]  | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.   |
| 42338 |          |  |
| 42339 | [EMLINK] | The link count of the parent directory would exceed {LINK_MAX}.  |

**mkdir()**

- 42340 [ENAMETOOLONG]  
 42341 The length of a component of a pathname is longer than {NAME\_MAX}.
- 42342 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 42343 directory or *path* is an empty string.
- 42344 [ENOSPC] The file system does not contain enough space to hold the contents of the new  
 42345 directory or to extend the parent directory of the new directory.
- 42346 [ENOTDIR] A component of the path prefix is not a directory.
- 42347 [EROFS] The parent directory resides on a read-only file system.
- 42348 In addition, the *mkdirat()* function shall fail if:
- 42349 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
 42350 neither AT\_FDCWD nor a valid file descriptor open for reading.
- 42351 These functions may fail if:
- 42352 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 42353 resolution of the *path* argument.
- 42354 [ENAMETOOLONG]  
 42355 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 42356 symbolic link produced an intermediate result with a length that exceeds  
 42357 {PATH\_MAX}.
- 42358 The *mkdirat()* function may fail if:
- 42359 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
 42360 file descriptor associated with a directory.

42361 **EXAMPLES**42362 **Creating a Directory**

42363 The following example shows how to create a directory named **/home/cnd/mod1**, with  
 42364 read/write/search permissions for owner and group, and with read/search permissions for  
 42365 others.

```
42366 #include <sys/types.h>
42367 #include <sys/stat.h>
42368 int status;
42369 ...
42370 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

42371 **APPLICATION USAGE**

42372 None.

42373 **RATIONALE**

42374 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

42375 4.3 BSD detects [ENAMETOOLONG].

42376 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the  
 42377 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2  
 42378 required that implementations provide a way to have the group ID be set to the group ID of the  
 42379 containing directory, but did not prohibit implementations also supporting a way to set the  
 42380 group ID to the effective group ID of the creating process. Conforming applications should not

42381 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
 42382 group ID after the directory is created, or determine under what conditions the implementation  
 42383 will set the desired group ID.

42384 The purpose of the *mkdirat()* function is to create a directory in directories other than the current  
 42385 working directory without exposure to race conditions. Any part of the path of a file could be  
 42386 changed in parallel to the call to *mkdir()*, resulting in unspecified behavior. By opening a file  
 42387 descriptor for the target directory and using the *mkdirat()* function it can be guaranteed that the  
 42388 newly created directory is located relative to the desired directory.

#### 42389 FUTURE DIRECTIONS

42390 None.

#### 42391 SEE ALSO

42392 *chmod()*, *mkdtemp()*, *mknod()*, *umask()*

42393 XBD [<sys/stat.h>](#), [<sys/types.h>](#)

#### 42394 CHANGE HISTORY

42395 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 42396 Issue 6

42397 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

42398 The following new requirements on POSIX implementations derive from alignment with the  
 42399 Single UNIX Specification:

- 42400 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 42401 required for conforming implementations of previous POSIX specifications, it was not  
 42402 required for UNIX applications.
- 42403 • The [ELOOP] mandatory error condition is added.
- 42404 • A second [ENAMETOOLONG] is added as an optional error condition.

42405 The following changes were made to align with the IEEE P1003.1a draft standard:

- 42406 • The [ELOOP] optional error condition is added.

#### 42407 Issue 7

42408 Austin Group Interpretation 1003.1-2001 #143 is applied.

42409 The *mkdirat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 42410 Set Part 2.

42411 Changes are made related to support for finegrained timestamps.

**mkdtemp()**42412 **NAME**

42413 mkdtemp, mkstemp — create a unique directory or file

42414 **SYNOPSIS**

```
42415 CX #include <stdlib.h>
42416 char *mkdtemp(char *template);
42417 int mkstemp(char *template);
```

42418 **DESCRIPTION**

42419 The *mkdtemp()* function uses the contents of *template* to construct a unique directory name. The  
 42420 string provided in *template* shall be a filename ending with six trailing 'X's. The *mkdtemp()*  
 42421 function shall replace each 'X' with a character from the portable filename character set. The  
 42422 characters are chosen such that the resulting name does not duplicate the name of an existing file  
 42423 at the time of a call to *mkdtemp()*. The unique directory name is used to attempt to create the  
 42424 directory using mode 0700 as modified by the file creation mask.

42425 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique  
 42426 filename, and return a file descriptor for the file open for reading and writing. The *mkstemp()*  
 42427 function shall create the file, and obtain a file descriptor for it, as if by a call to:

```
42428 open(filename, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)
```

42429 The function thus prevents any possible race condition between testing whether the file exists  
 42430 and opening it for use. The string in *template* should look like a filename with six trailing 'X's;  
 42431 *mkstemp()* replaces each 'X' with a character from the portable filename character set. The  
 42432 characters are chosen such that the resulting name does not duplicate the name of an existing file  
 42433 at the time of a call to *mkstemp()*.

42434 **RETURN VALUE**

42435 Upon successful completion, the *mkdtemp()* function shall return a pointer to the string  
 42436 containing the directory name if it was created. Otherwise, it shall return a null pointer and shall  
 42437 set *errno* to indicate the error.

42438 Upon successful completion, the *mkstemp()* function shall return an open file descriptor.  
 42439 Otherwise, it shall return `-1` if no suitable file could be created.

42440 **ERRORS**

42441 The *mkdtemp()* function shall fail if:

42442 [EACCES] Search permission is denied on a component of the path prefix, or write  
 42443 permission is denied on the parent directory of the directory to be created.

42444 [EINVAL] The string pointed to by *template* does not end in "XXXXXX".

42445 [ELOOP] A loop exists in symbolic links encountered during resolution of the path of  
 42446 the directory to be created.

42447 [EMLINK] The link count of the parent directory would exceed {LINK\_MAX}.

42448 [ENAMETOOLONG]

42449 The length of a component of a pathname is longer than {NAME\_MAX}.

42450 [ENOENT] A component of the path prefix specified by the *template* argument does not  
 42451 name an existing directory.

42452 [ENOSPC] The file system does not contain enough space to hold the contents of the new  
 42453 directory or to extend the parent directory of the new directory.

- 42454 [ENOTDIR] A component of the path prefix is not a directory.
- 42455 [EROFS] The parent directory resides on a read-only file system.
- 42456 The *mkdtemp()* function may fail if:
- 42457 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
42458 resolution of the path of the directory to be created.
- 42459 [ENAMETOOLONG]  
42460 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
42461 symbolic link produced an intermediate result with a length that exceeds  
42462 {PATH\_MAX}.
- 42463 The error conditions for the *mkstemp()* function are defined in *open()*.

**42464 EXAMPLES****42465 Generating a Filename**

42466 The following example creates a file with a 10-character name beginning with the characters  
42467 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file  
42468 descriptor that identifies the file.

```
42469 #include <stdlib.h>
42470 ...
42471 char template[] = "/tmp/fileXXXXXX";
42472 int fd;
42473
42474 fd = mkstemp(template);
```

**42474 APPLICATION USAGE**

42475 It is possible to run out of letters.

42476 The *mkdtemp()* and *mkstemp()* functions need not check to determine whether the filename part  
42477 of *template* exceeds the maximum allowable filename length.

**42478 RATIONALE**

42479 None.

**42480 FUTURE DIRECTIONS**

42481 None.

**42482 SEE ALSO**

42483 *getpid()*, *mkdir()*, *open()*, *tmpfile()*, *tmpnam()*

42484 XBD <stdlib.h>

**42485 CHANGE HISTORY**

42486 First released in Issue 4, Version 2.

**42487 Issue 5**

42488 Moved from X/OPEN UNIX extension to BASE.

**mkdtemp()**42489 **Issue 7**

42490 Austin Group Interpretation 1003.1-2001 #143 is applied.

42491 SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

42492 The *mkstemp()* function is moved from the XSI option to the Base.

42493 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API  
42494 Set Part 1.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42495 **NAME**

42496 mkfifo, mkfifoat — make a FIFO special file relative to directory file descriptor

42497 **SYNOPSIS**

42498 #include &lt;sys/stat.h&gt;

42499 int mkfifo(const char \*path, mode\_t mode);

42500 int mkfifoat(int fd, const char \*path, mode\_t mode);

42501 **DESCRIPTION**

42502 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by  
 42503 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission  
 42504 bits of the *mode* argument shall be modified by the process' file creation mask.

42505 When bits in *mode* other than the file permission bits are set, the effect is implementation-  
 42506 defined.

42507 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

42508 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set  
 42509 to the group ID of the parent directory or to the effective group ID of the process.  
 42510 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the  
 42511 parent directory. Implementations may, but need not, provide an implementation-defined way  
 42512 to initialize the FIFO's group ID to the effective group ID of the calling process.

42513 Upon successful completion, *mkfifo()* shall mark for update the last data access, last data  
 42514 modification, and last file status change timestamps of the file. Also, the last data modification  
 42515 and last file status change timestamps of the directory that contains the new entry shall be  
 42516 marked for update.

42517 The *mkfifoat()* function shall be equivalent to the *mkfifo()* function except in the case where *path*  
 42518 specifies a relative path. In this case the newly created FIFO is created relative to the directory  
 42519 associated with the file descriptor *fd* instead of the current working directory. If the file  
 42520 descriptor was opened without O\_SEARCH, the function shall check whether directory searches  
 42521 are permitted using the current permissions of the directory underlying the file descriptor. If the  
 42522 file descriptor was opened with O\_SEARCH, the function shall not perform the check.

42523 If *mkfifoat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 42524 directory is used and the behavior shall be identical to a call to *mkfifo()*.

42525 **RETURN VALUE**

42526 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 42527 return -1 and set *errno* to indicate the error. If -1 is returned, no FIFO shall be created.

42528 **ERRORS**

42529 These functions shall fail if:

42530 [EACCES] A component of the path prefix denies search permission, or write permission  
 42531 is denied on the parent directory of the FIFO to be created.

42532 [EEXIST] The named file already exists.

42533 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 42534 argument.

42535 [ENAMETOOLONG]

42536 The length of a component of a pathname is longer than {NAME\_MAX}.

**mkfifo()**

42537	[ENOENT]	A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string.
42538		
42539	[ENOSPC]	The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.
42540		
42541	[ENOTDIR]	A component of the path prefix is not a directory.
42542	[EROFS]	The named file resides on a read-only file system.
42543		The <i>mkfifoat()</i> function shall fail if:
42544	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
42545		
42546	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
42547		
42548		These functions may fail if:
42549	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
42550		
42551	[ENAMETOOLONG]	
42552		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
42553		
42554		
42555		The <i>mkfifoat()</i> function may fail if:
42556	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
42557		

42558 **EXAMPLES**42559 **Creating a FIFO File**

42560 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with  
 42561 read/write permissions for owner, and with read permissions for group and others.

```
42562 #include <sys/types.h>
42563 #include <sys/stat.h>
42564
42565 int status;
42566 ...
42567 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
42568 S_IRGRP | S_IROTH);
```

42568 **APPLICATION USAGE**

42569 None.

42570 **RATIONALE**

42571 The syntax of this function is intended to maintain compatibility with historical  
 42572 implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard  
 42573 but only for use in creating FIFO special files. The *mknod()* function was originally excluded  
 42574 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and  
 42575 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX  
 42576 Specification.

42577 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the  
 42578 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2

42579 required that implementations provide a way to have the group ID be set to the group ID of the  
 42580 containing directory, but did not prohibit implementations also supporting a way to set the  
 42581 group ID to the effective group ID of the creating process. Conforming applications should not  
 42582 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
 42583 group ID after the FIFO is created, or determine under what conditions the implementation will  
 42584 set the desired group ID.

42585 The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the  
 42586 current working directory without exposure to race conditions. Any part of the path of a file  
 42587 could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a  
 42588 file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that  
 42589 the newly created FIFO is located relative to the desired directory.

#### 42590 FUTURE DIRECTIONS

42591 None.

#### 42592 SEE ALSO

42593 *chmod()*, *mknod()*, *umask()*

42594 XBD [<sys/stat.h>](#), [<sys/types.h>](#)

#### 42595 CHANGE HISTORY

42596 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 42597 Issue 6

42598 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

42599 The following new requirements on POSIX implementations derive from alignment with the  
 42600 Single UNIX Specification:

- 42601 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
- 42602 required for conforming implementations of previous POSIX specifications, it was not
- 42603 required for UNIX applications.
- 42604 • The [ELOOP] mandatory error condition is added.
- 42605 • A second [ENAMETOOLONG] is added as an optional error condition.

42606 The following changes were made to align with the IEEE P1003.1a draft standard:

- 42607 • The [ELOOP] optional error condition is added.

#### 42608 Issue 7

42609 Austin Group Interpretation 1003.1-2001 #143 is applied.

42610 The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 42611 Set Part 2.

42612 Changes are made related to support for finegrained timestamps.

42613 Changes are made to allow a directory to be opened for searching.

# mknod()

42614 **NAME**

42615 mknod, mknodat — make directory, special file, or regular file

42616 **SYNOPSIS**

```
42617 xSI #include <sys/stat.h>
42618
42618 int mknod(const char *path, mode_t mode, dev_t dev);
42619 int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

42620 **DESCRIPTION**

42621 The *mknod()* function shall create a new file named by the pathname to which the argument *path*  
42622 points.

42623 The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the  
42624 following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

42631 The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S\_IFIFO or *dev* is  
42632 not 0, the behavior of *mknod()* is unspecified.

42633 The permissions for the new file are OR'ed into the *mode* argument, and may be selected from  
42634 any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

42651 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of  
42652 the file shall be initialized to either the effective group ID of the process or the group ID of the  
42653 parent directory. Implementations shall provide a way to initialize the file's group ID to the  
42654 group ID of the parent directory. Implementations may, but need not, provide an  
42655 implementation-defined way to initialize the file's group ID to the effective group ID of the  
42656 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file  
42657 mode creation mask of the process. The *mknod()* function shall clear each bit whose  
42658 corresponding bit in the file mode creation mask of the process is set.

- 42659 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].
- 42660 Upon successful completion, *mknod()* shall mark for update the last data access, last data  
42661 modification, and last file status change timestamps of the file. Also, the last data modification  
42662 and last file status change timestamps of the directory that contains the new entry shall be  
42663 marked for update.
- 42664 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO  
42665 special.
- 42666 The *mknodat()* function shall be equivalent to the *mknod()* function except in the case where *path*  
42667 specifies a relative path. In this case the newly created directory, special file, or regular file is  
42668 located relative to the directory associated with the file descriptor *fd* instead of the current  
42669 working directory. If the file descriptor was opened without O\_SEARCH, the function shall  
42670 check whether directory searches are permitted using the current permissions of the directory  
42671 underlying the file descriptor. If the file descriptor was opened with O\_SEARCH, the function  
42672 shall not perform the check.
- 42673 If *mknodat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
42674 directory is used and the behavior shall be identical to a call to *mknod()*.
- 42675 **RETURN VALUE**
- 42676 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
42677 return -1 and set *errno* to indicate the error. If -1 is returned, the new file shall not be created.
- 42678 **ERRORS**
- 42679 These functions shall fail if:
- |       |                |  |
|-------|----------------|--|
| 42680 | [EACCES]       | A component of the path prefix denies search permission, or write permission is denied on the parent directory.                |
| 42681 |                |  |
| 42682 | [EEXIST]       | The named file exists.   |
| 42683 | [EINVAL]       | An invalid argument exists.  |
| 42684 | [EIO]          | An I/O error occurred while accessing the file system.   |
| 42685 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                                     |
| 42686 |                |  |
| 42687 | [ENAMETOOLONG] |  |
| 42688 |                | The length of a component of a pathname is longer than {NAME_MAX}.   |
| 42689 | [ENOENT]       | A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string. |
| 42690 |                |  |
| 42691 | [ENOSPC]       | The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.       |
| 42692 |                |  |
| 42693 | [ENOTDIR]      | A component of the path prefix is not a directory.   |
| 42694 | [EPERM]        | The invoking process does not have appropriate privileges and the file type is not FIFO-special.                               |
| 42695 |                |  |
| 42696 | [EROFS]        | The directory in which the file is to be created is located on a read-only file system.  |
| 42697 |                |  |

**mknod()**

- 42698 The *mknodat()* function shall fail if:
- 42699 [EACCES] *fd* was not opened with O\_SEARCH and the permissions of the directory  
42700 underlying *fd* do not permit directory searches.
- 42701 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
42702 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.
- 42703 These functions may fail if:
- 42704 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
42705 resolution of the *path* argument.
- 42706 [ENAMETOOLONG]  
42707 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
42708 symbolic link produced an intermediate result with a length that exceeds  
42709 {PATH\_MAX}.
- 42710 The *mknodat()* function may fail if:
- 42711 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
42712 file descriptor associated with a directory.

42713 **EXAMPLES**42714 **Creating a FIFO Special File**

42715 The following example shows how to create a FIFO special file named */home/cnd/mod\_done*,  
42716 with read/write permissions for owner, and with read permissions for group and others.

```
42717 #include <sys/types.h>
42718 #include <sys/stat.h>
42719 dev_t dev;
42720 int status;
42721 ...
42722 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
42723 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

42724 **APPLICATION USAGE**

42725 The *mkfifo()* function is preferred over this function for making FIFO special files.

42726 **RATIONALE**

42727 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
42728 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
42729 that implementations provide a way to have the group ID be set to the group ID of the  
42730 containing directory, but did not prohibit implementations also supporting a way to set the  
42731 group ID to the effective group ID of the creating process. Conforming applications should not  
42732 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
42733 group ID after the file is created, or determine under what conditions the implementation will  
42734 set the desired group ID.

42735 The purpose of the *mknodat()* function is to create directories, special files, or regular files in  
42736 directories other than the current working directory without exposure to race conditions. Any  
42737 part of the path of a file could be changed in parallel to a call to *mknod()*, resulting in unspecified  
42738 behavior. By opening a file descriptor for the target directory and using the *mknodat()* function it  
42739 can be guaranteed that the newly created directory, special file, or regular file is located relative  
42740 to the desired directory.

42741 **FUTURE DIRECTIONS**

42742 None.

42743 **SEE ALSO**42744 *chmod()*, *creat()*, *exec*, *fstatat()*, *mkdir()*, *mkfifo()*, *open()*, *umask()*

42745 XBD &lt;sys/stat.h&gt;

42746 **CHANGE HISTORY**

42747 First released in Issue 4, Version 2.

42748 **Issue 5**

42749 Moved from X/OPEN UNIX extension to BASE.

42750 **Issue 6**

42751 The normative text is updated to avoid use of the term “must” for application requirements.

42752 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
42753 [ELOOP] error condition is added.42754 **Issue 7**

42755 Austin Group Interpretation 1003.1-2001 #143 is applied.

42756 The *mknodat()* function is added from The Open Group Technical Standard, 2006, Extended API  
42757 Set Part 2.

42758 Changes are made related to support for finegrained timestamps.

42759 Changes are made to allow a directory to be opened for searching.

**mkstemp()***System Interfaces*42760 **NAME**

42761 mkstemp — create a unique directory

42762 **SYNOPSIS**

```
42763 CX #include <stdlib.h>  
42764 int mkstemp(char *template);
```

42765 **DESCRIPTION**42766 Refer to *mkdtemp()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42767 **NAME**

42768 mktime — convert broken-down time into time since the Epoch

42769 **SYNOPSIS**

42770 #include &lt;time.h&gt;

42771 time\_t mktime(struct tm \*timeptr);

42772 **DESCRIPTION**

42773 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42774 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42775 volume of POSIX.1-2008 defers to the ISO C standard.

42776 The *mktime()* function shall convert the broken-down time, expressed as local time, in the  
 42777 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that  
 42778 of the values returned by *time()*. The original values of the *tm\_wday* and *tm\_yday* components of  
 42779 the structure are ignored, and the original values of the other components are not restricted to  
 42780 the ranges described in <time.h>.

42781 CX A positive or 0 value for *tm\_isdst* shall cause *mktime()* to presume initially that Daylight Savings  
 42782 Time, respectively, is or is not in effect for the specified time. A negative value for *tm\_isdst* shall  
 42783 cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the  
 42784 specified time.

42785 Local timezone information shall be set as though *mktime()* called *tzset()*.

42786 The relationship between the **tm** structure (defined in the <time.h> header) and the time in  
 42787 seconds since the Epoch is that the result shall be as specified in the expression given in the  
 42788 definition of seconds since the Epoch (see XBD Section 4.15, on page 113) corrected for timezone  
 42789 and any seasonal time adjustments, where the names in the structure and in the expression  
 42790 correspond.

42791 Upon successful completion, the values of the *tm\_wday* and *tm\_yday* components of the structure  
 42792 shall be set appropriately, and the other components are set to represent the specified time since  
 42793 the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final  
 42794 value of *tm\_mday* shall not be set until *tm\_mon* and *tm\_year* are determined.

42795 **RETURN VALUE**

42796 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type  
 42797 **time\_t**. If the time since the Epoch cannot be represented, the function shall return the value  
 42798 CX (**time\_t**)-1 and may set *errno* to indicate the error.

42799 **ERRORS**42800 The *mktime()* function may fail if:

42801 CX [EOVERFLOW] The result cannot be represented.

42802 **EXAMPLES**

42803 What day of the week is July 4, 2001?

42804 #include &lt;stdio.h&gt;

42805 #include &lt;time.h&gt;

42806 struct tm time\_str;

42807 char daybuf[20];

42808 int main(void)

42809 {

42810 time\_str.tm\_year = 2001 - 1900;

**mktime()**

```

42811         time_str.tm_mon = 7 - 1;
42812         time_str.tm_mday = 4;
42813         time_str.tm_hour = 0;
42814         time_str.tm_min = 0;
42815         time_str.tm_sec = 1;
42816         time_str.tm_isdst = -1;
42817         if (mktime(&time_str) == -1)
42818             (void)puts("-unknown-");
42819         else {
42820             (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
42821             (void)puts(daybuf);
42822         }
42823         return 0;
42824     }

```

42825 **APPLICATION USAGE**

42826 None.

42827 **RATIONALE**

42828 None.

42829 **FUTURE DIRECTIONS**

42830 None.

42831 **SEE ALSO**

42832 *asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), strftime(), strptime(), time(), tzset(),*  
42833 *utime()*

42834 XBD Section 4.15 (on page 113), **<time.h>**

42835 **CHANGE HISTORY**

42836 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C  
42837 standard.

42838 **Issue 6**

42839 Extensions beyond the ISO C standard are marked.

42840 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN  
42841 VALUE and ERRORS sections to add the optional [Eoverflow] error as a CX extension.

42842 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset()* function  
42843 to the SEE ALSO section.

42844 **NAME**42845 **mlock, munlock** — lock or unlock a range of process address space (**REALTIME**)42846 **SYNOPSIS**

```
42847 MLR #include <sys/mman.h>
42848 int mlock(const void *addr, size_t len);
42849 int munlock(const void *addr, size_t len);
```

42850 **DESCRIPTION**

42851 The *mlock()* function shall cause those whole pages containing any part of the address space of  
 42852 the process starting at address *addr* and continuing for *len* bytes to be memory resident until  
 42853 unlocked or until the process exits or *execs* another process image. The implementation may  
 42854 require that *addr* be a multiple of {PAGESIZE}.

42855 The *munlock()* function shall unlock those whole pages containing any part of the address space  
 42856 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many  
 42857 times *mlock()* has been called by the process for any of the pages in the specified range. The  
 42858 implementation may require that *addr* be a multiple of {PAGESIZE}.

42859 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address  
 42860 spaces of other processes, any locks established on those pages by another process are  
 42861 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a  
 42862 call to *munlock()* are also mapped into other portions of the address space of the calling process  
 42863 outside the range specified, any locks established on those pages via the other mappings are also  
 42864 unaffected by this call.

42865 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-  
 42866 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked  
 42867 with respect to the address space of the process. Memory residency of unlocked pages is  
 42868 unspecified.

42869 Appropriate privileges are required to lock process memory with *mlock()*.

42870 **RETURN VALUE**

42871 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.  
 42872 Otherwise, no change is made to any locks in the address space of the process, and the function  
 42873 shall return a value of -1 and set *errno* to indicate the error.

42874 **ERRORS**

42875 The *mlock()* and *munlock()* functions shall fail if:

42876 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does  
 42877 not correspond to valid mapped pages in the address space of the process.

42878 The *mlock()* function shall fail if:

42879 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 42880 when the call was made.

42881 The *mlock()* and *munlock()* functions may fail if:

42882 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

42883 The *mlock()* function may fail if:

42884 [ENOMEM] Locking the pages mapped by the specified range would exceed an  
 42885 implementation-defined limit on the amount of memory that the process may  
 42886 lock.

**mlock()**

System Interfaces

42887 [EPERM] The calling process does not have appropriate privileges to perform the  
42888 requested operation.

42889 **EXAMPLES**

42890 None.

42891 **APPLICATION USAGE**

42892 None.

42893 **RATIONALE**

42894 None.

42895 **FUTURE DIRECTIONS**

42896 None.

42897 **SEE ALSO**42898 *exec, exit(), fork(), mlockall(), munmap()*

42899 XBD &lt;sys/mman.h&gt;

42900 **CHANGE HISTORY**

42901 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

42902 **Issue 6**42903 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.42904 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42905 implementation does not support the Range Memory Locking option.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42906 **NAME**42907 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)42908 **SYNOPSIS**

```
42909 ML #include <sys/mman.h>
42910 int mlockall(int flags);
42911 int munlockall(void);
```

42912 **DESCRIPTION**

42913 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to  
 42914 be memory-resident until unlocked or until the process exits or *execs* another process image. The  
 42915 *flags* argument determines whether the pages to be locked are those currently mapped by the  
 42916 address space of the process, those that are mapped in the future, or both. The *flags* argument is  
 42917 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,  
 42918 defined in **<sys/mman.h>**:

42919 **MCL\_CURRENT** Lock all of the pages currently mapped into the address space of the process.

42920 **MCL\_FUTURE** Lock all of the pages that become mapped into the address space of the  
 42921 process in the future, when those mappings are established.

42922 If **MCL\_FUTURE** is specified, and the automatic locking of future mappings eventually causes  
 42923 the amount of locked memory to exceed the amount of available physical memory or any other  
 42924 implementation-defined limit, the behavior is implementation-defined. The manner in which the  
 42925 implementation informs the application of these situations is also implementation-defined.

42926 The *munlockall()* function shall unlock all currently mapped pages of the address space of the  
 42927 process. Any pages that become mapped into the address space of the process after a call to  
 42928 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying  
 42929 **MCL\_FUTURE** or a subsequent call to *mlockall()* specifying **MCL\_CURRENT**. If pages mapped  
 42930 into the address space of the process are also mapped into the address spaces of other processes  
 42931 and are locked by those processes, the locks established by the other processes shall be  
 42932 unaffected by a call by this process to *munlockall()*.

42933 Upon successful return from the *mlockall()* function that specifies **MCL\_CURRENT**, all currently  
 42934 mapped pages of the address space of the process shall be memory-resident and locked. Upon  
 42935 return from the *munlockall()* function, all currently mapped pages of the address space of the  
 42936 process shall be unlocked with respect to the address space of the process. The memory  
 42937 residency of unlocked pages is unspecified.

42938 Appropriate privileges are required to lock process memory with *mlockall()*.

42939 **RETURN VALUE**

42940 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no  
 42941 additional memory shall be locked, and the function shall return a value of  $-1$  and set *errno* to  
 42942 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address  
 42943 space is unspecified.

42944 If it is supported by the implementation, the *munlockall()* function shall always return a value of  
 42945 zero. Otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

42946 **ERRORS**

42947 The *mlockall()* function shall fail if:

42948 **[EAGAIN]** Some or all of the memory identified by the operation could not be locked  
 42949 when the call was made.

**mlockall()**

System Interfaces

- 42950 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.
- 42951 The *mlockall()* function may fail if:
- 42952 [ENOMEM] Locking all of the pages currently mapped into the address space of the  
42953 process would exceed an implementation-defined limit on the amount of  
42954 memory that the process may lock.
- 42955 [EPERM] The calling process does not have appropriate privileges to perform the  
42956 requested operation.
- 42957 **EXAMPLES**
- 42958 None.
- 42959 **APPLICATION USAGE**
- 42960 None.
- 42961 **RATIONALE**
- 42962 None.
- 42963 **FUTURE DIRECTIONS**
- 42964 None.
- 42965 **SEE ALSO**
- 42966 *exec*, *exit()*, *fork()*, *mlock()*, *munmap()*
- 42967 XBD <sys/mman.h>
- 42968 **CHANGE HISTORY**
- 42969 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 42970 **Issue 6**
- 42971 The *mlockall()* and *munlockall()* functions are marked as part of the Process Memory Locking  
42972 option.
- 42973 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42974 implementation does not support the Process Memory Locking option.

42975 **NAME**42976 `mmap` — map pages of memory42977 **SYNOPSIS**42978 `#include <sys/mman.h>`42979 `void *mmap(void *addr, size_t len, int prot, int flags,`  
42980 `int fildes, off_t off);`42981 **DESCRIPTION**42982 The `mmap()` function shall establish a mapping between an address space of a process and a  
42983 memory object.42984 The `mmap()` function shall be supported for the following memory objects:

- 42985 • Regular files
- 42986 SHM • Shared memory objects
- 42987 TYM • Typed memory objects

42988 Support for any other type of file is unspecified.

42989 The format of the call is as follows:

42990 `pa=mmap(addr, len, prot, flags, fildes, off);`

42991 The `mmap()` function shall establish a mapping between the address space of the process at an  
42992 address `pa` for `len` bytes to the memory object represented by the file descriptor `fildes` at offset `off`  
42993 for `len` bytes. The value of `pa` is an implementation-defined function of the parameter `addr` and  
42994 the values of `flags`, further described below. A successful `mmap()` call shall return `pa` as its result.  
42995 The address range starting at `pa` and continuing for `len` bytes shall be legitimate for the possible  
42996 (not necessarily current) address space of the process. The range of bytes starting at `off` and  
42997 continuing for `len` bytes shall be legitimate for the possible (not necessarily current) offsets in the  
42998 memory object represented by `fildes`.

42999 TYM If `fildes` represents a typed memory object opened with either the  
43000 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
43001 flag, the memory object to be mapped shall be that portion of the typed memory object allocated  
43002 by the implementation as specified below. In this case, if `off` is non-zero, the behavior of `mmap()`  
43003 is undefined. If `fildes` refers to a valid typed memory object that is not accessible from the calling  
43004 process, `mmap()` shall fail.

43005 The mapping established by `mmap()` shall replace any previous mappings for those whole pages  
43006 containing any part of the address space of the process starting at `pa` and continuing for `len`  
43007 bytes.

43008 If the size of the mapped file changes after the call to `mmap()` as a result of some other operation  
43009 on the mapped file, the effect of references to portions of the mapped region that correspond to  
43010 added or removed portions of the file is unspecified.

43011 If `len` is zero, `mmap()` shall fail and no mapping shall be established.

43012 The parameter `prot` determines whether read, write, execute, or some combination of accesses  
43013 are permitted to the data being mapped. The `prot` shall be either PROT\_NONE or the bitwise-  
43014 inclusive OR of one or more of the other flags in the following table, defined in the  
43015 `<sys/mman.h>` header.

**mmap()**

43016  
43017  
43018  
43019  
43020

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

43021  
43022

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap()* shall fail.

43023  
43024  
43025  
43026  
43027  
43028  
43029  
43030  
43031

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT\_WRITE has not been set and shall not permit any access where PROT\_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-inclusive OR of PROT\_READ and PROT\_WRITE. The file descriptor *filde*s shall have been opened with read permission, regardless of the protection options specified. If PROT\_WRITE is specified, the application shall ensure that it has opened the file descriptor *filde*s with write permission unless MAP\_PRIVATE is specified in the *flags* parameter as described below.

43032  
43033

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

43034  
43035  
43036  
43037

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

43038 XSI  
43039

It is implementation-defined whether MAP\_FIXED shall be supported. MAP\_FIXED shall be supported on XSI-conformant systems.

43040  
43041  
43042  
43043  
43044  
43045  
43046

MAP\_SHARED and MAP\_PRIVATE describe the disposition of write references to the memory object. If MAP\_SHARED is specified, write references shall change the underlying object. If MAP\_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP\_PRIVATE mapping is established are visible through the MAP\_PRIVATE mapping. Either MAP\_SHARED or MAP\_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

43047  
43048  
43049

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP\_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

43050 TYM  
43051  
43052  
43053  
43054  
43055  
43056  
43057  
43058  
43059  
43060  
43061  
43062

When *filde*s represents a typed memory object opened with either the POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *filde*s represents a typed memory object opened with the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *filde*s represents a typed memory object opened with the POSIX\_TYPED\_MEM\_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *filde*s represents a typed memory object opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag nor the POSIX\_TYPED\_MEM\_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two

- 43063 processes map an area of typed memory using the same *off* and *len* values and using file  
 43064 descriptors that refer to the same memory pool (either from the same port or from a different  
 43065 port), both processes shall map the same region of storage.
- 43066 When MAP\_FIXED is set in the *flags* argument, the implementation is informed that the value of  
 43067 *pa* shall be *addr*, exactly. If MAP\_FIXED is set, *mmap()* may return MAP\_FAILED and set *errno* to  
 43068 [EINVAL]. If a MAP\_FIXED request is successful, the mapping established by *mmap()* replaces  
 43069 any previous mappings for the pages in the range [*pa*,*pa+len*) of the process.
- 43070 When MAP\_FIXED is not set, the implementation uses *addr* in an implementation-defined  
 43071 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the  
 43072 implementation deems suitable for a mapping of *len* bytes to the file. All implementations  
 43073 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,  
 43074 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a  
 43075 process address near which the mapping should be placed. When the implementation selects a  
 43076 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.
- 43077 If MAP\_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off*  
 43078 parameter, modulo the page size as returned by *sysconf()* when passed \_SC\_PAGESIZE or  
 43079 \_SC\_PAGE\_SIZE. The implementation may require that *off* is a multiple of the page size. If  
 43080 MAP\_FIXED is specified, the implementation may require that *addr* is a multiple of the page  
 43081 size. The system performs mapping operations over whole pages. Thus, while the parameter *len*  
 43082 need not meet a size or alignment constraint, the system shall include, in any mapping  
 43083 operation, any partial page specified by the address range starting at *pa* and continuing for *len*  
 43084 bytes.
- 43085 The system shall always zero-fill any partial page at the end of an object. Further, the system  
 43086 shall never write out any modified portions of the last page of an object which are beyond its  
 43087 end. References within the address range starting at *pa* and continuing for *len* bytes to whole  
 43088 pages following the end of an object shall result in delivery of a SIGBUS signal.
- 43089 An implementation may generate SIGBUS signals when a reference would cause an error in the  
 43090 mapped object, such as out-of-space condition.
- 43091 The *mmap()* function shall add an extra reference to the file associated with the file descriptor  
 43092 *fd* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be  
 43093 removed when there are no more mappings to the file.
- 43094 The last data access timestamp of the mapped file may be marked for update at any time  
 43095 between the *mmap()* call and the corresponding *munmap()* call. The initial read or write  
 43096 reference to a mapped region shall cause the file's last data access timestamp to be marked for  
 43097 update if it has not already been marked for update.
- 43098 The last data modification and last file status change timestamps of a file that is mapped with  
 43099 MAP\_SHARED and PROT\_WRITE shall be marked for update at some point in the interval  
 43100 between a write reference to the mapped region and the next call to *msync()* with MS\_ASYNC or  
 43101 MS\_SYNC for that portion of the file by any process. If there is no such call and if the  
 43102 underlying file is modified as a result of a write reference, then these timestamps shall be  
 43103 marked for update at some time after the write reference.
- 43104 There may be implementation-defined limits on the number of memory regions that can be  
 43105 mapped (per process or per system).
- 43106 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a  
 43107 process is decreased by the use of *shmat()* is implementation-defined.
- 43108 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the

**mmap()**

43109 mappings in the address range starting at *addr* and continuing for *len* bytes may have been  
43110 unmapped.

**RETURN VALUE**

43112 Upon successful completion, the *mmap()* function shall return the address at which the mapping  
43113 was placed (*pa*); otherwise, it shall return a value of MAP\_FAILED and set *errno* to indicate the  
43114 error. The symbol MAP\_FAILED is defined in the `<sys/mman.h>` header. No successful return  
43115 from *mmap()* shall return the value MAP\_FAILED.

**ERRORS**

43116 The *mmap()* function shall fail if:

43118 [EACCES] The *fildev* argument is not open for read, regardless of the protection specified,  
43119 or *fildev* is not open for write and PROT\_WRITE was specified for a  
43120 MAP\_SHARED type mapping.

43121 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to  
43122 a lack of resources.

43123 [EBADF] The *fildev* argument is not a valid open file descriptor.

43124 [EINVAL] The value of *len* is zero.

43125 [EINVAL] The value of *flags* is invalid (neither MAP\_PRIVATE nor MAP\_SHARED is  
43126 set).

43127 [EMFILE] The number of mapped regions would exceed an implementation-defined  
43128 limit (per process or per system).

43129 [ENODEV] The *fildev* argument refers to a file whose type is not supported by *mmap()*.

43130 [ENOMEM] MAP\_FIXED was specified, and the range [*addr*,*addr+len*) exceeds that allowed  
43131 for the address space of a process; or, if MAP\_FIXED was not specified and  
43132 there is insufficient room in the address space to effect the mapping.

43133 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*,  
43134 because it would require more space than the system is able to supply.

43135 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory  
43136 object designated by *fildev* to allocate *len* bytes.

43137 [ENOTSUP] MAP\_FIXED or MAP\_PRIVATE was specified in the *flags* argument and the  
43138 implementation does not support this functionality.

43139 The implementation does not support the combination of accesses requested  
43140 in the *prot* argument.

43141 [ENXIO] Addresses in the range [*off*,*off+len*) are invalid for the object specified by *fildev*.

43142 [ENXIO] MAP\_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is  
43143 invalid for the object specified by *fildev*.

43144 TYM [ENXIO] The *fildev* argument refers to a typed memory object that is not accessible from  
43145 the calling process.

43146 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset  
43147 maximum established in the open file description associated with *fildev*.

43148 The *mmap()* function may fail if:

43149 [EINVAL] The *addr* argument (if MAP\_FIXED was specified) or *off* is not a multiple of the  
43150 page size as returned by *sysconf()*, or is considered invalid by the  
43151 implementation.

#### 43152 EXAMPLES

43153 None.

#### 43154 APPLICATION USAGE

43155 Use of *mmap()* may reduce the amount of memory available to other memory allocation  
43156 functions.

43157 Use of MAP\_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.  
43158 The use of MAP\_FIXED is discouraged, as it may prevent an implementation from making the  
43159 most effective use of resources. Most implementations require that *off* and *addr* are multiples of  
43160 the page size as returned by *sysconf()*.

43161 The application must ensure correct synchronization when using *mmap()* in conjunction with  
43162 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

43163 The *mmap()* function allows access to resources via address space manipulations, instead of  
43164 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the  
43165 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an  
43166 existing program might be changed to use *mmap()*, the following:

```
43167 fildes = open(...)
43168 lseek(fildes, some_offset)
43169 read(fildes, buf, len)
43170 /* Use data in buf. */
```

43171 becomes:

```
43172 fildes = open(...)
43173 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
43174 /* Use data at address. */
```

#### 43175 RATIONALE

43176 After considering several other alternatives, it was decided to adopt the *mmap()* definition  
43177 found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is  
43178 minimal, in that it describes only what has been built, and what appears to be necessary for a  
43179 general and portable mapping facility.

43180 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose  
43181 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,  
43182 and so on, into the address space of a process.

43183 When a mapping is established, it is possible that the implementation may need to map more  
43184 than is requested into the address space of the process because of hardware requirements. An  
43185 application, however, cannot count on this behavior. Implementations that do not use a paged  
43186 architecture may simply allocate a common memory region and return the address of it; such  
43187 implementations probably do not allocate any more than is necessary. References past the end of  
43188 the requested area are unspecified.

43189 If an application requests a mapping that would overlay existing mappings in the process, it  
43190 might be desirable that an implementation detect this and inform the application. However, the  
43191 default, portable (not MAP\_FIXED) operation does not overlay existing mappings. On the other  
43192 hand, if the program specifies a fixed address mapping (which requires some implementation

**mmap()**

43193 knowledge to determine a suitable address, if the function is supported at all), then the program  
43194 is presumed to be successfully managing its own address space and should be trusted when it  
43195 asks to map over existing data structures. Furthermore, it is also desirable to make as few system  
43196 calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()*  
43197 to the same address range. This volume of POSIX.1-2008 specifies that the new mappings  
43198 replace any existing mappings, following existing practice in this regard.

43199 It is not expected that all hardware implementations are able to support all combinations of  
43200 permissions at all addresses. Implementations are required to disallow write access to mappings  
43201 without write permission and to disallow access to mappings without any access permission.  
43202 Other than these restrictions, implementations may allow access types other than those  
43203 requested by the application. For example, if the application requests only `PROT_WRITE`, the  
43204 implementation may also allow read access. A call to *mmap()* fails if the implementation cannot  
43205 support allowing all the access requested by the application. For example, some  
43206 implementations cannot support a request for both write access and execute access  
43207 simultaneously. All implementations must support requests for no access, read access, write  
43208 access, and both read and write access. Strictly conforming code must only rely on the required  
43209 checks. These restrictions allow for portability across a wide range of hardware.

43210 The `MAP_FIXED` address treatment is likely to fail for non-page-aligned values and for certain  
43211 architecture-dependent address ranges. Conforming implementations cannot count on being  
43212 able to choose address values for `MAP_FIXED` without utilizing non-portable, implementation-  
43213 defined knowledge. Nonetheless, `MAP_FIXED` is provided as a standard interface conforming  
43214 to existing practice for utilizing such knowledge when it is available.

43215 Similarly, in order to allow implementations that do not support virtual addresses, support for  
43216 directly specifying any mapping addresses via `MAP_FIXED` is not required and thus a  
43217 conforming application may not count on it.

43218 The `MAP_PRIVATE` function can be implemented efficiently when memory protection hardware  
43219 is available. When such hardware is not available, implementations can implement such  
43220 “mappings” by simply making a real copy of the relevant data into process private memory,  
43221 though this tends to behave similarly to *read()*.

43222 The function has been defined to allow for many different models of using shared memory.  
43223 However, all uses are not equally portable across all machine architectures. In particular, the  
43224 *mmap()* function allows the system as well as the application to specify the address at which to  
43225 map a specific region of a memory object. The most portable way to use the function is always to  
43226 let the system choose the address, specifying `NULL` as the value for the argument *addr* and not  
43227 to specify `MAP_FIXED`.

43228 If it is intended that a particular region of a memory object be mapped at the same address in a  
43229 group of processes (on machines where this is even possible), then `MAP_FIXED` can be used to  
43230 pass in the desired mapping address. The system can still be used to choose the desired address  
43231 if the first such mapping is made without specifying `MAP_FIXED`, and then the resulting  
43232 mapping address can be passed to subsequent processes for them to pass in via `MAP_FIXED`.  
43233 The availability of a specific address range cannot be guaranteed, in general.

43234 The *mmap()* function can be used to map a region of memory that is larger than the current size  
43235 of the object. Memory access within the mapping but beyond the current end of the underlying  
43236 objects may result in `SIGBUS` signals being sent to the process. The reason for this is that the size  
43237 of the object can be manipulated by other processes and can change at any moment. The  
43238 implementation should tell the application that a memory reference is outside the object where  
43239 this can be detected; otherwise, written data may be lost and read data may not reflect actual  
43240 data in the object.

43241 Note that references beyond the end of the object do not extend the object as the new end cannot  
 43242 be determined precisely by most virtual memory hardware. Instead, the size can be directly  
 43243 manipulated by *ftruncate()*.

43244 Process memory locking does apply to shared memory regions, and the MEMLOCK\_FUTURE  
 43245 argument to *mlockall()* can be relied upon to cause new shared memory regions to be  
 43246 automatically locked.

43247 Existing implementations of *mmap()* return the value  $-1$  when unsuccessful. Since the casting of  
 43248 this value to type **void \*** cannot be guaranteed by the ISO C standard to be distinct from a  
 43249 successful value, this volume of POSIX.1-2008 defines the symbol MAP\_FAILED, which a  
 43250 conforming implementation does not return as the result of a successful call.

#### 43251 FUTURE DIRECTIONS

43252 None.

#### 43253 SEE ALSO

43254 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix\_typed\_mem\_open()*, *shmat()*,  
 43255 *sysconf()*

43256 XBD <sys/mman.h>

#### 43257 CHANGE HISTORY

43258 First released in Issue 4, Version 2.

#### 43259 Issue 5

43260 Moved from X/OPEN UNIX extension to BASE.

43261 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 43262 • The DESCRIPTION is extensively reworded.
- 43263 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 43264 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 43265 • The value returned on failure is the value of the constant MAP\_FAILED; this was  
 43266 previously defined as  $-1$ .

43267 Large File Summit extensions are added.

#### 43268 Issue 6

43269 The *mmap()* function is marked as part of the Memory Mapped Files option.

43270 The Open Group Corrigendum U028/6 is applied, changing (**void \***) $-1$  to MAP\_FAILED.

43271 The following new requirements on POSIX implementations derive from alignment with the  
 43272 Single UNIX Specification:

- 43273 • The DESCRIPTION is updated to describe the use of MAP\_FIXED.
- 43274 • The DESCRIPTION is updated to describe the addition of an extra reference to the file  
 43275 associated with the file descriptor passed to *mmap()*.
- 43276 • The DESCRIPTION is updated to state that there may be implementation-defined limits on  
 43277 the number of memory regions that can be mapped.
- 43278 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*  
 43279 argument.

**mmap()**

- 43280           • The [EINVAL] and [EMFILE] error conditions are added.
- 43281           • The [EOVERFLOW] error condition is added. This change is to support large files.
- 43282       The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 43283           • The DESCRIPTION is updated to describe the cases when MAP\_PRIVATE and  
43284           MAP\_FIXED need not be supported.
- 43285       The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 43286           • Semantics for typed memory objects are added to the DESCRIPTION.
- 43287           • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 43288           • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.
- 43289       The normative text is updated to avoid use of the term “must” for application requirements.
- 43290       IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code  
43291       in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).
- 43292       IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the  
43293       DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.
- 43294       **Issue 7**
- 43295       Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment  
43296       requirements and adding a note about the state of synchronization objects becoming undefined  
43297       when a shared region is unmapped.
- 43298       Functionality relating to the Memory Protection and Memory Mapped Files options is moved to  
43299       the Base.
- 43300       Changes are made related to support for finegrained timestamps.

43301 **NAME**

43302 modf, modff, modfl — decompose a floating-point number

43303 **SYNOPSIS**

43304 #include &lt;math.h&gt;

43305 double modf(double *x*, double \**iptr*);43306 float modff(float *value*, float \**iptr*);43307 long double modfl(long double *value*, long double \**iptr*);43308 **DESCRIPTION**

43309 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43310 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43311 volume of POSIX.1-2008 defers to the ISO C standard.

43312 These functions shall break the argument *x* into integral and fractional parts, each of which has  
 43313 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a  
 43314 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed  
 43315 to by *iptr*.

43316 **RETURN VALUE**43317 Upon successful completion, these functions shall return the signed fractional part of *x*.43318 **MX** If *x* is NaN, a NaN shall be returned, and \**iptr* shall be set to a NaN.43319 If *x* is ±Inf, ±0 shall be returned, and \**iptr* shall be set to ±Inf.43320 **ERRORS**

43321 No errors are defined.

43322 **EXAMPLES**

43323 None.

43324 **APPLICATION USAGE**43325 The *modf()* function computes the function result and \**iptr* such that:43326 *a* = modf(*x*, *iptr*) ;43327 *x* == *a*+\**iptr* ;

43328 allowing for the usual floating-point inaccuracies.

43329 **RATIONALE**

43330 None.

43331 **FUTURE DIRECTIONS**

43332 None.

43333 **SEE ALSO**43334 *frexp()*, *isnan()*, *ldexp()*

43335 XBD &lt;math.h&gt;

43336 **CHANGE HISTORY**

43337 First released in Issue 1. Derived from Issue 1 of the SVID.

43338 **Issue 5**

43339 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 43340 text was previously published in the APPLICATION USAGE section.

**modf()**43341 **Issue 6**

43342 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999  
43343 standard.

43344 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43345 revised to align with the ISO/IEC 9899:1999 standard.

43346 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43347 marked.

43348 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example  
43349 in the APPLICATION USAGE section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

43350 **NAME**43351 `mprotect` — set protection of memory mapping43352 **SYNOPSIS**43353 `#include <sys/mman.h>`43354 `int mprotect(void *addr, size_t len, int prot);`43355 **DESCRIPTION**

43356 The `mprotect()` function shall change the access protections to be that specified by `prot` for those  
 43357 whole pages containing any part of the address space of the process starting at address `addr` and  
 43358 continuing for `len` bytes. The parameter `prot` determines whether read, write, execute or some  
 43359 combination of accesses are permitted to the data being mapped. The `prot` argument should be  
 43360 either `PROT_NONE` or the bitwise-inclusive OR of one or more of `PROT_READ`, `PROT_WRITE`,  
 43361 and `PROT_EXEC`.

43362 If an implementation cannot support the combination of access types specified by `prot`, the call to  
 43363 `mprotect()` shall fail.

43364 An implementation may permit accesses other than those specified by `prot`; however, no  
 43365 implementation shall permit a write to succeed where `PROT_WRITE` has not been set or shall  
 43366 permit any access where `PROT_NONE` alone has been set. Implementations shall support at  
 43367 least the following values of `prot`: `PROT_NONE`, `PROT_READ`, `PROT_WRITE`, and the bitwise-  
 43368 inclusive OR of `PROT_READ` and `PROT_WRITE`. If `PROT_WRITE` is specified, the application  
 43369 shall ensure that it has opened the mapped objects in the specified address range with write  
 43370 permission, unless `MAP_PRIVATE` was specified in the original mapping, regardless of whether  
 43371 the file descriptors used to map the objects have since been closed.

43372 The implementation may require that `addr` be a multiple of the page size as returned by  
 43373 `sysconf()`.

43374 The behavior of this function is unspecified if the mapping was not established by a call to  
 43375 `mmap()`.

43376 When `mprotect()` fails for reasons other than `[EINVAL]`, the protections on some of the pages in  
 43377 the range `[addr,addr+len)` may have been changed.

43378 **RETURN VALUE**

43379 Upon successful completion, `mprotect()` shall return 0; otherwise, it shall return `-1` and set `errno`  
 43380 to indicate the error.

43381 **ERRORS**

43382 The `mprotect()` function shall fail if:

43383 `[EACCESS]` The `prot` argument specifies a protection that violates the access permission the  
 43384 process has to the underlying memory object.

43385 `[EAGAIN]` The `prot` argument specifies `PROT_WRITE` over a `MAP_PRIVATE` mapping  
 43386 and there are insufficient memory resources to reserve for locking the private  
 43387 page.

43388 `[ENOMEM]` Addresses in the range `[addr,addr+len)` are invalid for the address space of a  
 43389 process, or specify one or more pages which are not mapped.

43390 `[ENOMEM]` The `prot` argument specifies `PROT_WRITE` on a `MAP_PRIVATE` mapping, and  
 43391 it would require more space than the system is able to supply for locking the  
 43392 private pages, if required.

**mprotect()**

43393	[ENOTSUP]	The implementation does not support the combination of accesses requested in the <i>prot</i> argument.
43394		
43395		The <i>mprotect()</i> function may fail if:
43396	[EINVAL]	The <i>addr</i> argument is not a multiple of the page size as returned by <i>sysconf()</i> .
43397	<b>EXAMPLES</b>	
43398		None.
43399	<b>APPLICATION USAGE</b>	
43400		Most implementations require that <i>addr</i> is a multiple of the page size as returned by <i>sysconf()</i> .
43401	<b>RATIONALE</b>	
43402		None.
43403	<b>FUTURE DIRECTIONS</b>	
43404		None.
43405	<b>SEE ALSO</b>	
43406		<i>mmap()</i> , <i>sysconf()</i>
43407		XBD <sys/mman.h>
43408	<b>CHANGE HISTORY</b>	
43409		First released in Issue 4, Version 2.
43410	<b>Issue 5</b>	
43411		Moved from X/OPEN UNIX extension to BASE.
43412		Aligned with <i>mprotect()</i> in the POSIX Realtime Extension as follows:
43413		• The DESCRIPTION is largely reworded.
43414		• [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
43415		• [EAGAIN] is moved from the optional to the mandatory error conditions.
43416	<b>Issue 6</b>	
43417		The <i>mprotect()</i> function is marked as part of the Memory Protection option.
43418		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
43419		
43420		• The DESCRIPTION is updated to state that implementations require <i>addr</i> to be a multiple of the page size as returned by <i>sysconf()</i> .
43421		
43422		• The [EINVAL] error condition is added.
43423		The normative text is updated to avoid use of the term “must” for application requirements.
43424	<b>Issue 7</b>	
43425		SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.
43426		Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.
43427		
43428		The <i>mprotect()</i> function is moved from the Memory Protection option to the Base.

43429 **NAME**43430 mq\_close — close a message queue (**REALTIME**)43431 **SYNOPSIS**

```
43432 MSG #include <mqueue.h>
43433 int mq_close(mqd_t mqdes);
```

43434 **DESCRIPTION**

43435 The *mq\_close()* function shall remove the association between the message queue descriptor,  
 43436 *mqdes*, and its message queue. The results of using this message queue descriptor after successful  
 43437 return from this *mq\_close()*, and until the return of this message queue descriptor from a  
 43438 subsequent *mq\_open()*, are undefined.

43439 If the process has successfully attached a notification request to the message queue via this  
 43440 *mqdes*, this attachment shall be removed, and the message queue is available for another process  
 43441 to attach for notification.

43442 **RETURN VALUE**

43443 Upon successful completion, the *mq\_close()* function shall return a value of zero; otherwise, the  
 43444 function shall return a value of -1 and set *errno* to indicate the error.

43445 **ERRORS**

43446 The *mq\_close()* function shall fail if:

43447 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

43448 **EXAMPLES**

43449 None.

43450 **APPLICATION USAGE**

43451 None.

43452 **RATIONALE**

43453 None.

43454 **FUTURE DIRECTIONS**

43455 None.

43456 **SEE ALSO**

43457 *mq\_open()*, *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

43458 XBD <mqueue.h>

43459 **CHANGE HISTORY**

43460 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43461 **Issue 6**

43462 The *mq\_close()* function is marked as part of the Message Passing option.

43463 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 43464 implementation does not support the Message Passing option.

**mq\_getattr()**43465 **NAME**43466 `mq_getattr` — get message queue attributes (**REALTIME**)43467 **SYNOPSIS**

```
43468 MSG #include <mqqueue.h>
43469 int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

43470 **DESCRIPTION**

43471 The `mq_getattr()` function shall obtain status information and attributes of the message queue  
 43472 and the open message queue description associated with the message queue descriptor.

43473 The `mqdes` argument specifies a message queue descriptor.

43474 The results shall be returned in the `mq_attr` structure referenced by the `mqstat` argument.

43475 Upon return, the following members shall have the values associated with the open message  
 43476 queue description as set when the message queue was opened and as modified by subsequent  
 43477 `mq_setattr()` calls: `mq_flags`.

43478 The following attributes of the message queue shall be returned as set at message queue  
 43479 creation: `mq_maxmsg`, `mq_msgsize`.

43480 Upon return, the following members within the `mq_attr` structure referenced by the `mqstat`  
 43481 argument shall be set to the current state of the message queue:

43482 `mq_curmsgs` The number of messages currently on the queue.

43483 **RETURN VALUE**

43484 Upon successful completion, the `mq_getattr()` function shall return zero. Otherwise, the function  
 43485 shall return `-1` and set `errno` to indicate the error.

43486 **ERRORS**

43487 The `mq_getattr()` function may fail if:

43488 [EBADF] The `mqdes` argument is not a valid message queue descriptor.

43489 **EXAMPLES**

43490 See `mq_notify()`.

43491 **APPLICATION USAGE**

43492 None.

43493 **RATIONALE**

43494 None.

43495 **FUTURE DIRECTIONS**

43496 None.

43497 **SEE ALSO**

43498 `mq_notify()`, `mq_open()`, `mq_send()`, `mq_setattr()`, `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

43499 XBD `<mqqueue.h>`

43500 **CHANGE HISTORY**

43501 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43502 **Issue 6**

43503 The `mq_getattr()` function is marked as part of the Message Passing option.

43504 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 43505 implementation does not support the Message Passing option.

- 43506 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std  
43507 1003.1d-1999.
- 43508 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS  
43509 section to change the [EBADF] error from mandatory to optional.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**mq\_notify()**43510 **NAME**43511 mq\_notify — notify process that a message is available (**REALTIME**)43512 **SYNOPSIS**

```
43513 MSG #include <mqueue.h>
43514 int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

43515 **DESCRIPTION**

43516 If the argument *notification* is not NULL, this function shall register the calling process to be  
 43517 notified of message arrival at an empty message queue associated with the specified message  
 43518 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to  
 43519 the process when the message queue transitions from empty to non-empty. At any time, only  
 43520 one process may be registered for notification by a message queue. If the calling process or any  
 43521 other process has already registered for notification of message arrival at the specified message  
 43522 queue, subsequent attempts to register for that message queue shall fail.

43523 If *notification* is NULL and the process is currently registered for notification by the specified  
 43524 message queue, the existing registration shall be removed.

43525 When the notification is sent to the registered process, its registration shall be removed. The  
 43526 message queue shall then be available for registration.

43527 If a process has registered for notification of message arrival at a message queue and some  
 43528 thread is blocked in *mq\_receive()* or *mq\_timedreceive()* waiting to receive a message when a  
 43529 message arrives at the queue, the arriving message shall satisfy the appropriate *mq\_receive()* or  
 43530 *mq\_timedreceive()*, respectively. The resulting behavior is as if the message queue remains empty,  
 43531 and no notification shall be sent.

43532 **RETURN VALUE**

43533 Upon successful completion, the *mq\_notify()* function shall return a value of zero; otherwise, the  
 43534 function shall return a value of -1 and set *errno* to indicate the error.

43535 **ERRORS**

43536 The *mq\_notify()* function shall fail if:

43537 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

43538 [EBUSY] A process is already registered for notification by the message queue.

43539 The *mq\_notify()* function may fail if:

43540 [EINVAL] The *notification* argument is NULL and the process is currently not registered.

43541 **EXAMPLES**

43542 The following program registers a notification request for the message queue named in its  
 43543 command-line argument. Notification is performed by creating a thread. The thread executes a  
 43544 function which reads one message from the queue and then terminates the process.

```
43545 #include <pthread.h>
43546 #include <mqueue.h>
43547 #include <assert.h>
43548 #include <stdio.h>
43549 #include <stdlib.h>
43550 #include <unistd.h>

43551 static void /* Thread start function */
43552 tfunc(union sigval sv)
43553 {
```

```

43554     struct mq_attr attr;
43555     ssize_t nr;
43556     void *buf;
43557     mqd_t mqdes = *((mqd_t *) sv.sival_ptr);
43558
43558     /* Determine maximum msg size; allocate buffer to receive msg */
43559     if (mq_getattr(mqdes, &attr) == -1) {
43560         perror("mq_getattr");
43561         exit(EXIT_FAILURE);
43562     }
43563     buf = malloc(attr.mq_msgsize);
43564
43564     if (buf == NULL) {
43565         perror("malloc");
43566         exit(EXIT_FAILURE);
43567     }
43568
43568     nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
43569     if (nr == -1) {
43570         perror("mq_receive");
43571         exit(EXIT_FAILURE);
43572     }
43573
43573     printf("Read %ld bytes from message queue\n", (long) nr);
43574     free(buf);
43575     exit(EXIT_SUCCESS);          /* Terminate the process */
43576 }
43577
43577 int
43578 main(int argc, char *argv[])
43579 {
43580     mqd_t mqdes;
43581     struct sigevent not;
43582
43582     assert(argc == 2);
43583
43583     mqdes = mq_open(argv[1], O_RDONLY);
43584     if (mqdes == (mqd_t) -1) {
43585         perror("mq_open");
43586         exit(EXIT_FAILURE);
43587     }
43588
43588     not.sigev_notify = SIGEV_THREAD;
43589     not.sigev_notify_function = tfunc;
43590     not.sigev_notify_attributes = NULL;
43591     not.sigev_value.sival_ptr = &mqdes;    /* Arg. to thread func. */
43592     if (mq_notify(mqdes, &not) == -1) {
43593         perror("mq_notify");
43594         exit(EXIT_FAILURE);
43595     }
43596
43596     pause();    /* Process will be terminated by thread function */
43597 }

```

**mq\_notify()**43598 **APPLICATION USAGE**

43599 None.

43600 **RATIONALE**

43601 None.

43602 **FUTURE DIRECTIONS**

43603 None.

43604 **SEE ALSO**43605 *mq\_open()*, *mq\_send()*, *mq\_receive()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

43606 XBD &lt;mqqueue.h&gt;

43607 **CHANGE HISTORY**

43608 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43609 **Issue 6**43610 The *mq\_notify()* function is marked as part of the Message Passing option.43611 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
43612 implementation does not support the Message Passing option.43613 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std  
43614 1003.1d-1999.43615 **Issue 7**43616 SD5-XSH-ERN-38 is applied, adding the *mq\_timedreceive()* function to the DESCRIPTION.

43617 Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

43618 An example is added.

43619 **NAME**43620 mq\_open — open a message queue (**REALTIME**)43621 **SYNOPSIS**

```
43622 MSG #include <mqueue.h>
43623 mqd_t mq_open(const char *name, int oflag, ...);
```

43624 **DESCRIPTION**

43625 The *mq\_open()* function shall establish the connection between a process and a message queue  
 43626 with a message queue descriptor. It shall create an open message queue description that refers to  
 43627 the message queue, and a message queue descriptor that refers to that open message queue  
 43628 description. The message queue descriptor is used by other functions to refer to that message  
 43629 queue. The *name* argument points to a string naming a message queue. It is unspecified whether  
 43630 the name appears in the file system and is visible to other functions that take pathnames as  
 43631 arguments. The *name* argument conforms to the construction rules for a pathname, except that  
 43632 the interpretation of <slash> characters other than the leading <slash> character in *name* is  
 43633 implementation-defined, and that the length limits for the *name* argument are implementation-  
 43634 defined and need not be the same as the pathname limits {PATH\_MAX} and {NAME\_MAX}. If  
 43635 *name* begins with the <slash> character, then processes calling *mq\_open()* with the same value of  
 43636 *name* shall refer to the same message queue object, as long as that name has not been removed. If  
 43637 *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name*  
 43638 argument is not the name of an existing message queue and creation is not requested, *mq\_open()*  
 43639 shall fail and return an error.

43640 A message queue descriptor may be implemented using a file descriptor, in which case  
 43641 applications can open up to at least {OPEN\_MAX} file and message queues.

43642 The *oflag* argument requests the desired receive and/or send access to the message queue. The  
 43643 requested access permission to receive messages or send messages shall be granted if the calling  
 43644 process would be granted read or write access, respectively, to an equivalently protected file.

43645 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications  
 43646 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

43647 **O\_RDONLY** Open the message queue for receiving messages. The process can use the  
 43648 returned message queue descriptor with *mq\_receive()*, but not *mq\_send()*. A  
 43649 message queue may be open multiple times in the same or different processes  
 43650 for receiving messages.

43651 **O\_WRONLY** Open the queue for sending messages. The process can use the returned  
 43652 message queue descriptor with *mq\_send()* but not *mq\_receive()*. A message  
 43653 queue may be open multiple times in the same or different processes for  
 43654 sending messages.

43655 **O\_RDWR** Open the queue for both receiving and sending messages. The process can use  
 43656 any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**. A message  
 43657 queue may be open multiple times in the same or different processes for  
 43658 sending messages.

43659 Any combination of the remaining flags may be specified in the value of *oflag*:

43660 **O\_CREAT** Create a message queue. It requires two additional arguments: *mode*, which  
 43661 shall be of type **mode\_t**, and *attr*, which shall be a pointer to an **mq\_attr**  
 43662 structure. If the pathname *name* has already been used to create a message  
 43663 queue that still exists, then this flag shall have no effect, except as noted under  
 43664 **O\_EXCL**. Otherwise, a message queue shall be created without any messages

**mq\_open()**

43665		in it. The user ID of the message queue shall be set to the effective user ID of the process. The group ID of the message queue shall be set to the effective group ID of the process; however, if the <i>name</i> argument is visible in the file system, the group ID may be set to the group ID of the containing directory. When bits in <i>mode</i> other than the file permission bits are specified, the effect is unspecified. If <i>attr</i> is NULL, the message queue shall be created with implementation-defined default message queue attributes. If <i>attr</i> is non-NULL and the calling process has appropriate privileges on <i>name</i> , the message queue <i>mq_maxmsg</i> and <i>mq_msgsize</i> attributes shall be set to the values of the corresponding members in the <b>mq_attr</b> structure referred to by <i>attr</i> . If <i>attr</i> is non-NULL, but the calling process does not have appropriate privileges on <i>name</i> , the <i>mq_open()</i> function shall fail and return an error without creating the message queue.
43666		
43667		
43668		
43669		
43670		
43671		
43672		
43673		
43674		
43675		
43676		
43677		
43678	O_EXCL	If O_EXCL and O_CREAT are set, <i>mq_open()</i> shall fail if the message queue <i>name</i> exists. The check for the existence of the message queue and the creation of the message queue if it does not exist shall be atomic with respect to other threads executing <i>mq_open()</i> naming the same <i>name</i> with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.
43679		
43680		
43681		
43682		
43683		
43684	O_NONBLOCK	Determines whether an <i>mq_send()</i> or <i>mq_receive()</i> waits for resources or messages that are not currently available, or fails with <i>errno</i> set to [EAGAIN]; see <i>mq_send()</i> and <i>mq_receive()</i> for details.
43685		
43686		
43687		The <i>mq_open()</i> function does not add or remove messages from the queue.
43688	<b>RETURN VALUE</b>	
43689		Upon successful completion, the function shall return a message queue descriptor; otherwise, the function shall return ( <b>mqd_t</b> )-1 and set <i>errno</i> to indicate the error.
43690		
43691	<b>ERRORS</b>	
43692		The <i>mq_open()</i> function shall fail if:
43693	[EACCES]	The message queue exists and the permissions specified by <i>oflag</i> are denied, or the message queue does not exist and permission to create the message queue is denied.
43694		
43695		
43696	[EEXIST]	O_CREAT and O_EXCL are set and the named message queue already exists.
43697	[EINTR]	The <i>mq_open()</i> function was interrupted by a signal.
43698	[EINVAL]	The <i>mq_open()</i> function is not supported for the given name.
43699	[EINVAL]	O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL, and either <i>mq_maxmsg</i> or <i>mq_msgsize</i> was less than or equal to zero.
43700		
43701	[EMFILE]	Too many message queue descriptors or file descriptors are currently in use by this process.
43702		
43703	[ENFILE]	Too many message queues are currently open in the system.
43704	[ENOENT]	O_CREAT is not set and the named message queue does not exist.
43705	[ENOSPC]	There is insufficient space for the creation of the new message queue.

43706 If any of the following conditions occur, the *mq\_open()* function may return (**mqd\_t**)-1 and set  
43707 *errno* to the corresponding value.

43708 [ENAMETOOLONG]

43709 The length of the *name* argument exceeds {\_POSIX\_PATH\_MAX} on systems  
43710 XSI that do not support the XSI option or exceeds {\_XOPEN\_PATH\_MAX} on XSI  
43711 systems, or has a pathname component that is longer than  
43712 XSI {\_POSIX\_NAME\_MAX} on systems that do not support the XSI option or  
43713 longer than {\_XOPEN\_NAME\_MAX} on XSI systems.

#### 43714 EXAMPLES

43715 None.

#### 43716 APPLICATION USAGE

43717 None.

#### 43718 RATIONALE

43719 None.

#### 43720 FUTURE DIRECTIONS

43721 A future version might require the *mq\_open()* and *mq\_unlink()* functions to have semantics  
43722 similar to normal file system operations.

#### 43723 SEE ALSO

43724 *mq\_close()*, *mq\_getattr()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgctl()*, *msgget()*,  
43725 *msgrcv()*, *msgsnd()*

43726 XBD <mqqueue.h>

#### 43727 CHANGE HISTORY

43728 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 43729 Issue 6

43730 The *mq\_open()* function is marked as part of the Message Passing option.

43731 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
43732 implementation does not support the Message Passing option.

43733 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added to the SEE ALSO section for  
43734 alignment with IEEE Std 1003.1d-1999.

43735 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

43736 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of  
43737 the permission bits in the DESCRIPTION. The change is made for consistency with the  
43738 *shm\_open()* and *sem\_open()* functions.

#### 43739 Issue 7

43740 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and  
43741 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

43742 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

43743 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
43744 user ID and group ID of the message queue.

**mq\_receive()**43745 **NAME**43746 `mq_receive`, `mq_timedreceive` — receive a message from a message queue (**REALTIME**)43747 **SYNOPSIS**

```

43748 MSG #include <mqqueue.h>
43749
43749 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
43750                 unsigned *msg_prio);
43751
43751 #include <mqqueue.h>
43752 #include <time.h>
43753
43753 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
43754                       size_t msg_len, unsigned *restrict msg_prio,
43755                       const struct timespec *restrict abstime);

```

43756 **DESCRIPTION**

43757 The `mq_receive()` function shall receive the oldest of the highest priority message(s) from the  
 43758 message queue specified by `mqdes`. If the size of the buffer in bytes, specified by the `msg_len`  
 43759 argument, is less than the `mq_msgsize` attribute of the message queue, the function shall fail and  
 43760 return an error. Otherwise, the selected message shall be removed from the queue and copied to  
 43761 the buffer pointed to by the `msg_ptr` argument.

43762 If the value of `msg_len` is greater than {SSIZE\_MAX}, the result is implementation-defined.

43763 If the argument `msg_prio` is not NULL, the priority of the selected message shall be stored in the  
 43764 location referenced by `msg_prio`.

43765 If the specified message queue is empty and O\_NONBLOCK is not set in the message queue  
 43766 description associated with `mqdes`, `mq_receive()` shall block until a message is enqueued on the  
 43767 message queue or until `mq_receive()` is interrupted by a signal. If more than one thread is waiting  
 43768 to receive a message when a message arrives at an empty queue and the Priority Scheduling  
 43769 option is supported, then the thread of highest priority that has been waiting the longest shall be  
 43770 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the  
 43771 message. If the specified message queue is empty and O\_NONBLOCK is set in the message  
 43772 queue description associated with `mqdes`, no message shall be removed from the queue, and  
 43773 `mq_receive()` shall return an error.

43774 The `mq_timedreceive()` function shall receive the oldest of the highest priority messages from the  
 43775 message queue specified by `mqdes` as described for the `mq_receive()` function. However, if  
 43776 O\_NONBLOCK was not specified when the message queue was opened via the `mq_open()`  
 43777 function, and no message exists on the queue to satisfy the receive, the wait for such a message  
 43778 shall be terminated when the specified timeout expires. If O\_NONBLOCK is set, this function is  
 43779 equivalent to `mq_receive()`.

43780 The timeout expires when the absolute time specified by `abstime` passes, as measured by the  
 43781 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 43782 `abstime`), or if the absolute time specified by `abstime` has already been passed at the time of the  
 43783 call.

43784 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
 43785 be the resolution of the clock on which it is based. The `timespec` argument is defined in the  
 43786 `<time.h>` header.

43787 Under no circumstance shall the operation fail with a timeout if a message can be removed from  
 43788 the message queue immediately. The validity of the `abstime` parameter need not be checked if a  
 43789 message can be removed from the message queue immediately.

43790 **RETURN VALUE**

43791 Upon successful completion, the *mq\_receive()* and *mq\_timedreceive()* functions shall return the  
 43792 length of the selected message in bytes and the message shall be removed from the queue.  
 43793 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1,  
 43794 and set *errno* to indicate the error.

43795 **ERRORS**

43796 These functions shall fail if:

- 43797 [EAGAIN] O\_NONBLOCK was set in the message description associated with *mqdes*, and  
 43798 the specified message queue is empty.
- 43799 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.
- 43800 [EMSGSIZE] The specified message buffer size, *msg\_len*, is less than the message size  
 43801 attribute of the message queue.
- 43802 [EINTR] The *mq\_receive()* or *mq\_timedreceive()* operation was interrupted by a signal.
- 43803 [EINVAL] The process or thread would have blocked, and the *abstime* parameter  
 43804 specified a nanoseconds field value less than zero or greater than or equal to  
 43805 1 000 million.
- 43806 [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 43807 but no message arrived on the queue before the specified timeout expired.

43808 These functions may fail if:

- 43809 [EBADMSG] The implementation has detected a data corruption problem with the  
 43810 message.

43811 **EXAMPLES**

43812 None.

43813 **APPLICATION USAGE**

43814 None.

43815 **RATIONALE**

43816 None.

43817 **FUTURE DIRECTIONS**

43818 None.

43819 **SEE ALSO**

43820 *mq\_open()*, *mq\_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*

43821 XBD [<mqqueue.h>](#), [<time.h>](#)

43822 **CHANGE HISTORY**

43823 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43824 **Issue 6**

43825 The *mq\_receive()* function is marked as part of the Message Passing option.

43826 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to  
 43827 *msg\_len* rather than *maxsize*.

43828 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 43829 implementation does not support the Message Passing option.

43830 The following new requirements on POSIX implementations derive from alignment with the  
 43831 Single UNIX Specification:

**mq\_receive()**

- 43832                   • In this function it is possible for the return value to exceed the range of the type **ssize\_t**  
43833                   (since **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the  
43834                   size of the **size\_t** object is added to the description to resolve this conflict.
- 43835                   The *mq\_timedreceive()* function is added for alignment with IEEE Std 1003.1d-1999.
- 43836                   The **restrict** keyword is added to the *mq\_timedreceive()* prototype for alignment with the  
43837                   ISO/IEC 9899: 1999 standard.
- 43838                   IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq\_timedreceive()*  
43839                   from **int** to **ssize\_t**.
- 43840   **Issue 7**
- 43841                   The *mq\_timedreceive()* function is moved from the Timeouts option to the Base.
- 43842                   Functionality relating to the Timers option is moved to the Base.

## 43843 NAME

43844 mq\_send, mq\_timedsend — send a message to a message queue (REALTIME)

## 43845 SYNOPSIS

```

43846 MSG #include <mqueue.h>
43847 int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43848             unsigned msg_prio);
43849 #include <mqueue.h>
43850 #include <time.h>
43851 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43852                 unsigned msg_prio, const struct timespec *abstime);

```

## 43853 DESCRIPTION

43854 The *mq\_send()* function shall add the message pointed to by the argument *msg\_ptr* to the  
 43855 message queue specified by *mqdes*. The *msg\_len* argument specifies the length of the message, in  
 43856 bytes, pointed to by *msg\_ptr*. The value of *msg\_len* shall be less than or equal to the *mq\_msgsize*  
 43857 attribute of the message queue, or *mq\_send()* shall fail.

43858 If the specified message queue is not full, *mq\_send()* shall behave as if the message is inserted  
 43859 into the message queue at the position indicated by the *msg\_prio* argument. A message with a  
 43860 larger numeric value of *msg\_prio* shall be inserted before messages with lower values of  
 43861 *msg\_prio*. A message shall be inserted after other messages in the queue, if any, with equal  
 43862 *msg\_prio*. The value of *msg\_prio* shall be less than {MQ\_PRIO\_MAX}.

43863 If the specified message queue is full and O\_NONBLOCK is not set in the message queue  
 43864 description associated with *mqdes*, *mq\_send()* shall block until space becomes available to  
 43865 enqueue the message, or until *mq\_send()* is interrupted by a signal. If more than one thread is  
 43866 waiting to send when space becomes available in the message queue and the Priority Scheduling  
 43867 option is supported, then the thread of the highest priority that has been waiting the longest  
 43868 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is  
 43869 unblocked. If the specified message queue is full and O\_NONBLOCK is set in the message  
 43870 queue description associated with *mqdes*, the message shall not be queued and *mq\_send()* shall  
 43871 return an error.

43872 The *mq\_timedsend()* function shall add a message to the message queue specified by *mqdes* in the  
 43873 manner defined for the *mq\_send()* function. However, if the specified message queue is full and  
 43874 O\_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for  
 43875 sufficient room in the queue shall be terminated when the specified timeout expires. If  
 43876 O\_NONBLOCK is set in the message queue description, this function shall be equivalent to  
 43877 *mq\_send()*.

43878 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the  
 43879 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 43880 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
 43881 call.

43882 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
 43883 be the resolution of the clock on which it is based. The *timespec* argument is defined in the  
 43884 **<time.h>** header.

43885 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the  
 43886 queue to add the message immediately. The validity of the *abstime* parameter need not be  
 43887 checked when there is sufficient room in the queue.

**mq\_send()**43888 **RETURN VALUE**

43889 Upon successful completion, the *mq\_send()* and *mq\_timedsend()* functions shall return a value of  
 43890 zero. Otherwise, no message shall be enqueued, the functions shall return *-1*, and *errno* shall be  
 43891 set to indicate the error.

43892 **ERRORS**

43893 The *mq\_send()* and *mq\_timedsend()* functions shall fail if:

- 43894 [EAGAIN] The O\_NONBLOCK flag is set in the message queue description associated  
 43895 with *mqdes*, and the specified message queue is full.
- 43896 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.
- 43897 [EINTR] A signal interrupted the call to *mq\_send()* or *mq\_timedsend()*.
- 43898 [EINVAL] The value of *msg\_prio* was outside the valid range.
- 43899 [EINVAL] The process or thread would have blocked, and the *abstime* parameter  
 43900 specified a nanoseconds field value less than zero or greater than or equal to  
 43901 1 000 million.
- 43902 [EMSGSIZE] The specified message length, *msg\_len*, exceeds the message size attribute of  
 43903 the message queue.
- 43904 [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 43905 but the timeout expired before the message could be added to the queue.

43906 **EXAMPLES**

43907 None.

43908 **APPLICATION USAGE**

43909 The value of the symbol {MQ\_PRIO\_MAX} limits the number of priority levels supported by the  
 43910 application. Message priorities range from 0 to {MQ\_PRIO\_MAX}-1.

43911 **RATIONALE**

43912 None.

43913 **FUTURE DIRECTIONS**

43914 None.

43915 **SEE ALSO**

43916 *mq\_open()*, *mq\_receive()*, *mq\_setattr()*, *time()*

43917 XBD <mqqueue.h>, <time.h>

43918 **CHANGE HISTORY**

43919 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43920 **Issue 6**

43921 The *mq\_send()* function is marked as part of the Message Passing option.

43922 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 43923 implementation does not support the Message Passing option.

43924 The *mq\_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

43925 **Issue 7**

43926 The *mq\_timedsend()* function is moved from the Timeouts option to the Base.

43927 Functionality relating to the Timers option is moved to the Base.

43928 **NAME**43929 mq\_setattr — set message queue attributes (**REALTIME**)43930 **SYNOPSIS**

```
43931 MSG #include <mqqueue.h>
43932 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
43933 struct mq_attr *restrict omqstat);
```

43934 **DESCRIPTION**

43935 The *mq\_setattr()* function shall set attributes associated with the open message queue  
 43936 description referenced by the message queue descriptor specified by *mqdes*.

43937 The message queue attributes corresponding to the following members defined in the **mq\_attr**  
 43938 structure shall be set to the specified values upon successful completion of *mq\_setattr()*:

43939 *mq\_flags* The value of this member is the bitwise-logical OR of zero or more of  
 43940 O\_NONBLOCK and any implementation-defined flags.

43941 The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the **mq\_attr** structure  
 43942 shall be ignored by *mq\_setattr()*.

43943 If *omqstat* is non-NULL, the *mq\_setattr()* function shall store, in the location referenced by  
 43944 *omqstat*, the previous message queue attributes and the current queue status. These values shall  
 43945 be the same as would be returned by a call to *mq\_getattr()* at that point.

43946 **RETURN VALUE**

43947 Upon successful completion, the function shall return a value of zero and the attributes of the  
 43948 message queue shall have been changed as specified.

43949 Otherwise, the message queue attributes shall be unchanged, and the function shall return a  
 43950 value of -1 and set *errno* to indicate the error.

43951 **ERRORS**

43952 The *mq\_setattr()* function shall fail if:

43953 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

43954 **EXAMPLES**

43955 None.

43956 **APPLICATION USAGE**

43957 None.

43958 **RATIONALE**

43959 None.

43960 **FUTURE DIRECTIONS**

43961 None.

43962 **SEE ALSO**

43963 *mq\_open()*, *mq\_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

43964 XBD **<mqqueue.h>**

43965 **CHANGE HISTORY**

43966 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**mq\_setattr()**43967 **Issue 6**

43968 The *mq\_setattr()* function is marked as part of the Message Passing option.

43969 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
43970 implementation does not support the Message Passing option.

43971 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std  
43972 1003.1d-1999.

43973 The **restrict** keyword is added to the *mq\_setattr()* prototype for alignment with the  
43974 ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

43975 **NAME**43976 `mq_timedreceive` — receive a message from a message queue (**ADVANCED REALTIME**)43977 **SYNOPSIS**

```
43978 MSG #include <mqueue.h>
43979 #include <time.h>
43980
43980 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
43981 size_t msg_len, unsigned *restrict msg_prio,
43982 const struct timespec *restrict abstime);
```

43983 **DESCRIPTION**43984 Refer to [mq\\_receive\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**mq\_timedsend()**43985 **NAME**43986 `mq_timedsend` — send a message to a message queue (**ADVANCED REALTIME**)43987 **SYNOPSIS**

```
43988 MSG #include <mqueue.h>
43989 #include <time.h>
43990 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43991 unsigned msg_prio, const struct timespec *abstime);
```

43992 **DESCRIPTION**43993 Refer to [mq\\_send\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

43994 **NAME**43995 mq\_unlink — remove a message queue (**REALTIME**)43996 **SYNOPSIS**

```
43997 MSG #include <mqueue.h>
43998 int mq_unlink(const char *name);
```

43999 **DESCRIPTION**

44000 The *mq\_unlink()* function shall remove the message queue named by the string name. If one or  
 44001 more processes have the message queue open when *mq\_unlink()* is called, destruction of the  
 44002 message queue shall be postponed until all references to the message queue have been closed.  
 44003 However, the *mq\_unlink()* call need not block until all references have been closed; it may return  
 44004 immediately.

44005 After a successful call to *mq\_unlink()*, reuse of the name shall subsequently cause *mq\_open()* to  
 44006 behave as if no message queue of this name exists (that is, *mq\_open()* will fail if **O\_CREAT** is not  
 44007 set, or will create a new message queue if **O\_CREAT** is set).

44008 **RETURN VALUE**

44009 Upon successful completion, the function shall return a value of zero. Otherwise, the named  
 44010 message queue shall be unchanged by this function call, and the function shall return a value of  
 44011  $-1$  and set *errno* to indicate the error.

44012 **ERRORS**44013 The *mq\_unlink()* function shall fail if:

44014 [EACCES] Permission is denied to unlink the named message queue.

44015 [ENOENT] The named message queue does not exist.

44016 The *mq\_unlink()* function may fail if:

44017 [ENAMETOOLONG]

44018 The length of the *name* argument exceeds  $\{\_POSIX\_PATH\_MAX\}$  on systems  
 44019 XSI that do not support the XSI option or exceeds  $\{\_XOPEN\_PATH\_MAX\}$  on XSI  
 44020 systems, or has a pathname component that is longer than  
 44021 XSI  $\{\_POSIX\_NAME\_MAX\}$  on systems that do not support the XSI option or  
 44022 longer than  $\{\_XOPEN\_NAME\_MAX\}$  on XSI systems. A call to *mq\_unlink()*  
 44023 with a *name* argument that contains the same message queue name as was  
 44024 previously used in a successful *mq\_open()* call shall not give an  
 44025 [ENAMETOOLONG] error.

44026 **EXAMPLES**

44027 None.

44028 **APPLICATION USAGE**

44029 None.

44030 **RATIONALE**

44031 None.

44032 **FUTURE DIRECTIONS**

44033 A future version might require the *mq\_open()* and *mq\_unlink()* functions to have semantics  
 44034 similar to normal file system operations.

**mq\_unlink()**44035 **SEE ALSO**44036 *mq\_close()*, *mq\_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*44037 XBD <[mqqueue.h](#)>44038 **CHANGE HISTORY**

44039 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44040 **Issue 6**44041 The *mq\_unlink()* function is marked as part of the Message Passing option.

44042 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, the named message queue is unchanged by this function.

44044 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

44046 **Issue 7**

44047 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error .

44049 Austin Group Interpretation 1003.1-2001 #141 is applied.

*System Interfaces***rand48()**44050 **NAME**

44051           rand48 — generate uniformly distributed pseudo-random signed long integers

44052 **SYNOPSIS**44053 XSI       #include <stdlib.h>  
44054           long rand48(void);44055 **DESCRIPTION**44056           Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**msgctl()**44057 **NAME**

44058 msgctl — XSI message control operations

44059 **SYNOPSIS**

```
44060 XSI #include <sys/msg.h>
44061 int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

44062 **DESCRIPTION**

44063 The *msgctl()* function operates on XSI message queues (see XBD Section 3.224, on page 69). It is  
 44064 unspecified whether this function interoperates with the realtime interprocess communication  
 44065 facilities defined in Section 2.8 (on page 497).

44066 The *msgctl()* function shall provide message control operations as specified by *cmd*. The  
 44067 following values for *cmd*, and the message control operations they specify, are:

44068 **IPC\_STAT** Place the current value of each member of the **msqid\_ds** data structure  
 44069 associated with *msqid* into the structure pointed to by *buf*. The contents of this  
 44070 structure are defined in **<sys/msg.h>**.

44071 **IPC\_SET** Set the value of the following members of the **msqid\_ds** data structure  
 44072 associated with *msqid* to the corresponding value found in the structure  
 44073 pointed to by *buf*:

```
44074 msg_perm.uid
44075 msg_perm.gid
44076 msg_perm.mode
44077 msg_qbytes
```

44078 **IPC\_SET** can only be executed by a process with appropriate privileges or that  
 44079 has an effective user ID equal to the value of **msg\_perm.cuid** or  
 44080 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*. Only a  
 44081 process with appropriate privileges can raise the value of **msg\_qbytes**.

44082 **IPC\_RMID** Remove the message queue identifier specified by *msqid* from the system and  
 44083 destroy the message queue and **msqid\_ds** data structure associated with it.  
 44084 **IPC\_RMID** can only be executed by a process with appropriate privileges or  
 44085 one that has an effective user ID equal to the value of **msg\_perm.cuid** or  
 44086 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*.

44087 **RETURN VALUE**

44088 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 44089 indicate the error.

44090 **ERRORS**

44091 The *msgctl()* function shall fail if:

44092 **[EACCES]** The argument *cmd* is **IPC\_STAT** and the calling process does not have read  
 44093 permission; see Section 2.7 (on page 496).

44094 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*  
 44095 is not a valid command.

44096 **[EPERM]** The argument *cmd* is **IPC\_RMID** or **IPC\_SET** and the effective user ID of the  
 44097 calling process is not equal to that of a process with appropriate privileges and  
 44098 it is not equal to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data  
 44099 structure associated with *msqid*.

44100 [EPERM] The argument *cmd* is IPC\_SET, an attempt is being made to increase to the  
 44101 value of **msg\_qbytes**, and the effective user ID of the calling process does not  
 44102 have appropriate privileges.

#### 44103 EXAMPLES

44104 None.

#### 44105 APPLICATION USAGE

44106 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
 44107 (IPC). Application developers who need to use IPC should design their applications so that  
 44108 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to  
 44109 use the alternative interfaces.

#### 44110 RATIONALE

44111 None.

#### 44112 FUTURE DIRECTIONS

44113 None.

#### 44114 SEE ALSO

44115 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
 44116 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

44117 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)

#### 44118 CHANGE HISTORY

44119 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 44120 Issue 5

44121 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 44122 DIRECTIONS to a new APPLICATION USAGE section.

**msgget()**44123 **NAME**44124 `msgget` — get the XSI message queue identifier44125 **SYNOPSIS**

```
44126 XSI #include <sys/msg.h>
44127 int msgget(key_t key, int msgflg);
```

44128 **DESCRIPTION**

44129 The `msgget()` function operates on XSI message queues (see XBD Section 3.224, on page 69). It is  
 44130 unspecified whether this function interoperates with the realtime interprocess communication  
 44131 facilities defined in Section 2.8 (on page 497).

44132 The `msgget()` function shall return the message queue identifier associated with the argument  
 44133 `key`.

44134 A message queue identifier, associated message queue, and data structure (see `<sys/msg.h>`),  
 44135 shall be created for the argument `key` if one of the following is true:

- 44136 • The argument `key` is equal to `IPC_PRIVATE`.
- 44137 • The argument `key` does not already have a message queue identifier associated with it, and  
 44138 (`msgflg & IPC_CREAT`) is non-zero.

44139 Upon creation, the data structure associated with the new message queue identifier shall be  
 44140 initialized as follows:

- 44141 • `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` shall be set equal to  
 44142 the effective user ID and effective group ID, respectively, of the calling process.
- 44143 • The low-order 9 bits of `msg_perm.mode` shall be set equal to the low-order 9 bits of `msgflg`.
- 44144 • `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` shall be set equal to 0.
- 44145 • `msg_ctime` shall be set equal to the current time.
- 44146 • `msg_qbytes` shall be set equal to the system limit.

44147 **RETURN VALUE**

44148 Upon successful completion, `msgget()` shall return a non-negative integer, namely a message  
 44149 queue identifier. Otherwise, it shall return `-1` and set `errno` to indicate the error.

44150 **ERRORS**

44151 The `msgget()` function shall fail if:

- |       |          |   |
|-------|----------|---|
| 44152 | [EACCES] | A message queue identifier exists for the argument <code>key</code> , but operation permission as specified by the low-order 9 bits of <code>msgflg</code> would not be granted; see Section 2.7 (on page 496). |
| 44153 |          |   |
| 44154 |          |   |
| 44155 | [EEXIST] | A message queue identifier exists for the argument <code>key</code> but $((msgflg \& IPC\_CREAT) \&\& (msgflg \& IPC\_EXCL))$ is non-zero.  |
| 44156 |          |   |
| 44157 | [ENOENT] | A message queue identifier does not exist for the argument <code>key</code> and $(msgflg \& IPC\_CREAT)$ is 0.  |
| 44158 |          |   |
| 44159 | [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.  |
| 44160 |          |   |
| 44161 |          |   |

44162 **EXAMPLES**

44163 None.

44164 **APPLICATION USAGE**

44165 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
 44166 (IPC). Application developers who need to use IPC should design their applications so that  
 44167 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to  
 44168 use the alternative interfaces.

44169 **RATIONALE**

44170 None.

44171 **FUTURE DIRECTIONS**

44172 None.

44173 **SEE ALSO**

44174 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
 44175 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

44176 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)44177 **CHANGE HISTORY**

44178 First released in Issue 2. Derived from Issue 2 of the SVID.

44179 **Issue 5**

44180 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 44181 DIRECTIONS to a new APPLICATION USAGE section.

**msgrcv()**44182 **NAME**44183 `msgrcv` — XSI message receive operation44184 **SYNOPSIS**

```
44185 XSI #include <sys/msg.h>
44186
44186 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
44187               int msgflg);
```

44188 **DESCRIPTION**

44189 The `msgrcv()` function operates on XSI message queues (see XBD Section 3.224, on page 69). It is  
 44190 unspecified whether this function interoperates with the realtime interprocess communication  
 44191 facilities defined in Section 2.8 (on page 497).

44192 The `msgrcv()` function shall read a message from the queue associated with the message queue  
 44193 identifier specified by `msqid` and place it in the user-defined buffer pointed to by `msgp`.

44194 The application shall ensure that the argument `msgp` points to a user-defined buffer that contains  
 44195 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 44196 the data bytes of the message. The structure below is an example of what this user-defined  
 44197 buffer might look like:

```
44198 struct mymsg {
44199     long    mtype;    /* Message type. */
44200     char    mtext[1]; /* Message text. */
44201 }
```

44202 The structure member `mtype` is the received message's type as specified by the sending process.

44203 The structure member `mtext` is the text of the message.

44204 The argument `msgsz` specifies the size in bytes of `mtext`. The received message shall be truncated  
 44205 to `msgsz` bytes if it is larger than `msgsz` and (`msgflg & MSG_NOERROR`) is non-zero. The  
 44206 truncated part of the message shall be lost and no indication of the truncation shall be given to  
 44207 the calling process.

44208 If the value of `msgsz` is greater than `{SSIZE_MAX}`, the result is implementation-defined.

44209 The argument `msgtyp` specifies the type of message requested as follows:

- 44210 • If `msgtyp` is 0, the first message on the queue shall be received.
- 44211 • If `msgtyp` is greater than 0, the first message of type `msgtyp` shall be received.
- 44212 • If `msgtyp` is less than 0, the first message of the lowest type that is less than or equal to the  
 44213 absolute value of `msgtyp` shall be received.

44214 The argument `msgflg` specifies the action to be taken if a message of the desired type is not on the  
 44215 queue. These are as follows:

- 44216 • If (`msgflg & IPC_NOWAIT`) is non-zero, the calling thread shall return immediately with a  
 44217 return value of `-1` and `errno` set to `[ENOMSG]`.
- 44218 • If (`msgflg & IPC_NOWAIT`) is 0, the calling thread shall suspend execution until one of the  
 44219 following occurs:
  - 44220 — A message of the desired type is placed on the queue.
  - 44221 — The message queue identifier `msqid` is removed from the system; when this occurs,  
 44222 `errno` shall be set equal to `[EIDRM]` and `-1` shall be returned.

44223 — The calling thread receives a signal that is to be caught; in this case a message is not  
 44224 received and the calling thread resumes execution in the manner prescribed in  
 44225 *sigaction()*.

44226 Upon successful completion, the following actions are taken with respect to the data structure  
 44227 associated with *msqid*:

- 44228 • **msg\_qnum** shall be decremented by 1.
- 44229 • **msg\_lrpid** shall be set equal to the process ID of the calling process.
- 44230 • **msg\_rtime** shall be set equal to the current time.

#### 44231 RETURN VALUE

44232 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually  
 44233 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return  
 44234 (**ssize\_t**)-1, and *errno* shall be set to indicate the error.

#### 44235 ERRORS

44236 The *msgrcv()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 44237 | [E2BIG]  | The value of <i>mtext</i> is greater than <i>msgsz</i> and ( <i>msgflg</i> & MSG_NOERROR) is 0.         |
| 44238 | [EACCES] | Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 496).   |
| 44239 |          |   |
| 44240 | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                                   |
| 44241 | [EINTR]  | The <i>msgrcv()</i> function was interrupted by a signal.   |
| 44242 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier.   |
| 44243 | [ENOMSG] | The queue does not contain a message of the desired type and ( <i>msgflg</i> & IPC_NOWAIT) is non-zero. |
| 44244 |          |   |

#### 44245 EXAMPLES

##### 44246 Receiving a Message

44247 The following example receives the first message on the queue (based on the value of the *msgtyp*  
 44248 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has  
 44249 previously been set). This call specifies that an error should be reported if no message is  
 44250 available, but not if the message is too large. The message size is calculated directly using the  
 44251 *sizeof* operator.

```

44252 #include <sys/msg.h>
44253 ...
44254 int result;
44255 int msqid;
44256 struct message {
44257     long type;
44258     char text[20];
44259 } msg;
44260 long msgtyp = 0;
44261 ...
44262 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
44263               msgtyp, MSG_NOERROR | IPC_NOWAIT);
  
```

**msgrcv()**44264 **APPLICATION USAGE**

44265 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
 44266 (IPC). Application developers who need to use IPC should design their applications so that  
 44267 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to  
 44268 use the alternative interfaces.

44269 **RATIONALE**

44270 None.

44271 **FUTURE DIRECTIONS**

44272 None.

44273 **SEE ALSO**

44274 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
 44275 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgsnd\(\)](#),  
 44276 [sigaction\(\)](#)

44277 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)

44278 **CHANGE HISTORY**

44279 First released in Issue 2. Derived from Issue 2 of the SVID.

44280 **Issue 5**

44281 The type of the return value is changed from **int** to **ssize\_t**, and a warning is added to the  
 44282 DESCRIPTION about values of *msgsz* larger the {SSIZE\_MAX}.

44283 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 44284 DIRECTIONS to the APPLICATION USAGE section.

44285 **Issue 6**

44286 The normative text is updated to avoid use of the term “must” for application requirements.

44287 **NAME**

44288 msgsnd — XSI message send operation

44289 **SYNOPSIS**

```
44290 XSI #include <sys/msg.h>
44291 int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

44292 **DESCRIPTION**

44293 The *msgsnd()* function operates on XSI message queues (see XBD Section 3.224, on page 69). It is  
 44294 unspecified whether this function interoperates with the realtime interprocess communication  
 44295 facilities defined in Section 2.8 (on page 497).

44296 The *msgsnd()* function shall send a message to the queue associated with the message queue  
 44297 identifier specified by *msqid*.

44298 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
 44299 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 44300 the data bytes of the message. The structure below is an example of what this user-defined  
 44301 buffer might look like:

```
44302 struct mymsg {
44303     long    mtype;           /* Message type. */
44304     char    mtext[1];      /* Message text. */
44305 }
```

44306 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving  
 44307 process for message selection.

44308 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range  
 44309 from 0 to a system-imposed maximum.

44310 The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- 44311 • The number of bytes already on the queue is equal to **msg\_qbytes**; see `<sys/msg.h>`.
- 44312 • The total number of messages on all queues system-wide is equal to the system-imposed  
 44313 limit.

44314 These actions are as follows:

- 44315 • If (*msgflg* & `IPC_NOWAIT`) is non-zero, the message shall not be sent and the calling  
 44316 thread shall return immediately.
- 44317 • If (*msgflg* & `IPC_NOWAIT`) is 0, the calling thread shall suspend execution until one of the  
 44318 following occurs:
  - 44319 — The condition responsible for the suspension no longer exists, in which case the  
 44320 message is sent.
  - 44321 — The message queue identifier *msqid* is removed from the system; when this occurs,  
 44322 *errno* shall be set equal to `[EIDRM]` and `-1` shall be returned.
  - 44323 — The calling thread receives a signal that is to be caught; in this case the message is not  
 44324 sent and the calling thread resumes execution in the manner prescribed in *sigaction()*.

**msgsnd()**

44325 Upon successful completion, the following actions are taken with respect to the data structure  
44326 associated with *msqid*; see `<sys/msg.h>`:

- 44327 • **msg\_qnum** shall be incremented by 1.
- 44328 • **msg\_lspid** shall be set equal to the process ID of the calling process.
- 44329 • **msg\_stime** shall be set equal to the current time.

**RETURN VALUE**

44331 Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,  
44332 *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

**ERRORS**

44334 The *msgsnd()* function shall fail if:

- 44335 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page  
44336 496).
- 44337 [EAGAIN] The message cannot be sent for one of the reasons cited above and (*msgflg* &  
44338 IPC\_NOWAIT) is non-zero.
- 44339 [EIDRM] The message queue identifier *msqid* is removed from the system.
- 44340 [EINTR] The *msgsnd()* function was interrupted by a signal.
- 44341 [EINVAL] The value of *msqid* is not a valid message queue identifier, or the value of  
44342 *mtype* is less than 1; or the value of *msgsz* is less than 0 or greater than the  
44343 system-imposed limit.

**EXAMPLES****Sending a Message**

44345 The following example sends a message to the queue identified by the *msqid* argument  
44346 (assuming that value has previously been set). This call specifies that an error should be  
44347 reported if no message is available. The message size is calculated directly using the *sizeof*  
44348 operator.  
44349

```
44350 #include <sys/msg.h>
44351 ...
44352 int result;
44353 int msqid;
44354 struct message {
44355     long type;
44356     char text[20];
44357 } msg;
44358 msg.type = 1;
44359 strcpy(msg.text, "This is message 1");
44360 ...
44361 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

**APPLICATION USAGE**

44362 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
44363 (IPC). Application developers who need to use IPC should design their applications so that  
44364 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to  
44365 use the alternative interfaces.  
44366

44367 **RATIONALE**

44368 None.

44369 **FUTURE DIRECTIONS**

44370 None.

44371 **SEE ALSO**

44372 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
 44373 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#),  
 44374 [sigaction\(\)](#)

44375 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)44376 **CHANGE HISTORY**

44377 First released in Issue 2. Derived from Issue 2 of the SVID.

44378 **Issue 5**

44379 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 44380 DIRECTIONS to a new APPLICATION USAGE section.

44381 **Issue 6**

44382 The normative text is updated to avoid use of the term “must” for application requirements.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

# msync()

44383 **NAME**

44384 `msync` — synchronize memory with physical storage

44385 **SYNOPSIS**

```
44386 XSI|SIO #include <sys/mman.h>
44387 int msync(void *addr, size_t len, int flags);
```

44388 **DESCRIPTION**

44389 The `msync()` function shall write all modified data to permanent storage locations, if any, in  
 44390 those whole pages containing any part of the address space of the process starting at address  
 44391 `addr` and continuing for `len` bytes. If no such storage exists, `msync()` need not have any effect. If  
 44392 requested, the `msync()` function shall then invalidate cached copies of data.

44393 The implementation may require that `addr` be a multiple of the page size as returned by  
 44394 `sysconf()`.

44395 For mappings to files, the `msync()` function shall ensure that all write operations are completed  
 44396 as defined for synchronized I/O data integrity completion. It is unspecified whether the  
 44397 implementation also writes out other file attributes. When the `msync()` function is called on  
 44398 MAP\_PRIVATE mappings, any modified data shall not be written to the underlying object and  
 44399 shall not cause such data to be made visible to other processes. It is unspecified whether data in  
 44400 SHM|TYM MAP\_PRIVATE mappings has any permanent storage locations. The effect of `msync()` on a  
 44401 shared memory object or a typed memory object is unspecified. The behavior of this function is  
 44402 unspecified if the mapping was not established by a call to `mmap()`.

44403 The `flags` argument is constructed from the bitwise-inclusive OR of one or more of the following  
 44404 flags defined in the `<sys/mman.h>` header:

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

44409 When MS\_ASYNC is specified, `msync()` shall return immediately once all the write operations  
 44410 are initiated or queued for servicing; when MS\_SYNC is specified, `msync()` shall not return until  
 44411 all write operations are completed as defined for synchronized I/O data integrity completion.  
 44412 Either MS\_ASYNC or MS\_SYNC shall be specified, but not both.

44413 When MS\_INVALIDATE is specified, `msync()` shall invalidate all cached copies of mapped data  
 44414 that are inconsistent with the permanent storage locations such that subsequent references shall  
 44415 obtain data that was consistent with the permanent storage locations sometime between the call  
 44416 to `msync()` and the first subsequent memory reference to the data.

44417 If `msync()` causes any write to a file, the file's last data modification and last file status change  
 44418 timestamps shall be marked for update.

44419 **RETURN VALUE**

44420 Upon successful completion, `msync()` shall return 0; otherwise, it shall return -1 and set `errno` to  
 44421 indicate the error.

44422 **ERRORS**

44423 The `msync()` function shall fail if:

44424 [EBUSY] Some or all of the addresses in the range starting at `addr` and continuing for `len`  
 44425 bytes are locked, and MS\_INVALIDATE is specified.

- 44426 [EINVAL] The value of *flags* is invalid.
- 44427 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are  
44428 outside the range allowed for the address space of a process or specify one or  
44429 more pages that are not mapped.

44430 The *msync()* function may fail if:

- 44431 [EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf()*.

#### 44432 EXAMPLES

44433 None.

#### 44434 APPLICATION USAGE

44435 The *msync()* function is only supported if the Synchronized Input and Output option is  
44436 supported, and thus need not be available on all implementations.

44437 The *msync()* function should be used by programs that require a memory object to be in a  
44438 known state; for example, in building transaction facilities.

44439 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees  
44440 that *msync()* is the only control over when pages are or are not written to disk.

#### 44441 RATIONALE

44442 The *msync()* function writes out data in a mapped region to the permanent storage for the  
44443 underlying object. The call to *msync()* ensures data integrity of the file.

44444 After the data is written out, any cached data may be invalidated if the MS\_INVALIDATE flag  
44445 was specified. This is useful on systems that do not support read/write consistency.

#### 44446 FUTURE DIRECTIONS

44447 None.

#### 44448 SEE ALSO

44449 *mmap()*, *sysconf()*

44450 XBD <[sys/mman.h](#)>

#### 44451 CHANGE HISTORY

44452 First released in Issue 4, Version 2.

#### 44453 Issue 5

44454 Moved from X/OPEN UNIX extension to BASE.

44455 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 44456 • The DESCRIPTION is extensively reworded.
- 44457 • [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

#### 44458 Issue 6

44459 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input  
44460 and Output options.

44461 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 44462 • The [EBUSY] mandatory error condition is added.

44463 The following new requirements on POSIX implementations derive from alignment with the  
44464 Single UNIX Specification:

**msync()**

- 44465           • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple  
44466           of the page size.
- 44467           • The second [EINVAL] error condition is made mandatory.
- 44468           The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to  
44469           typed memory objects.
- 44470   **Issue 7**
- 44471           Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment  
44472           requirements.
- 44473           SD5-XSH-ERN-110 is applied.
- 44474           The *msync()* function is marked as part of the Synchronized Input and Output option or XSI  
44475           option as the Memory Mapped Files is moved to the Base.
- 44476           Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

44477 **NAME**

44478           munlock — unlock a range of process address space

44479 **SYNOPSIS**

44480 MLR       #include &lt;sys/mman.h&gt;

44481           int munlock(const void \*addr, size\_t len);

44482 **DESCRIPTION**44483           Refer to *mlock()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**munlockall()***System Interfaces*44484 **NAME**

44485           munlockall — unlock the address space of a process

44486 **SYNOPSIS**44487 ML       #include <sys/mman.h>  
44488           int munlockall(void);44489 **DESCRIPTION**44490           Refer to *mlockall()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

44491 **NAME**44492 `munmap` — unmap pages of memory44493 **SYNOPSIS**44494 `#include <sys/mman.h>`44495 `int munmap(void *addr, size_t len);`44496 **DESCRIPTION**

44497 The `munmap()` function shall remove any mappings for those entire pages containing any part of  
 44498 the address space of the process starting at `addr` and continuing for `len` bytes. Further references  
 44499 to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no  
 44500 mappings in the specified address range, then `munmap()` has no effect.

44501 The implementation may require that `addr` be a multiple of the page size as returned by  
 44502 `sysconf()`.

44503 If a mapping to be removed was private, any modifications made in this address range shall be  
 44504 discarded.

44505 ML|MLR Any memory locks (see `mlock()` and `mlockall()`) associated with this address range shall be  
 44506 removed, as if by an appropriate call to `munlock()`.

44507 TYM If a mapping removed from a typed memory object causes the corresponding address range of  
 44508 the memory pool to be inaccessible by any process in the system except through allocatable  
 44509 mappings (that is, mappings of typed memory objects opened with the  
 44510 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag), then that range of the memory pool shall  
 44511 become deallocated and may become available to satisfy future typed memory allocation  
 44512 requests.

44513 A mapping removed from a typed memory object opened with the  
 44514 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag shall not affect in any way the availability of  
 44515 that typed memory for allocation.

44516 The behavior of this function is unspecified if the mapping was not established by a call to  
 44517 `mmap()`.

44518 **RETURN VALUE**

44519 Upon successful completion, `munmap()` shall return 0; otherwise, it shall return -1 and set `errno`  
 44520 to indicate the error.

44521 **ERRORS**

44522 The `munmap()` function shall fail if:

44523 [EINVAL] Addresses in the range `[addr,addr+len)` are outside the valid range for the  
 44524 address space of a process.

44525 [EINVAL] The `len` argument is 0.

44526 The `munmap()` function may fail if:

44527 [EINVAL] The `addr` argument is not a multiple of the page size as returned by `sysconf()`.

**munmap()**44528 **EXAMPLES**

44529 None.

44530 **APPLICATION USAGE**

44531 None.

44532 **RATIONALE**44533 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

44534 It is possible that an application has applied process memory locking to a region that contains  
 44535 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,  
 44536 causes those locks to be removed.

44537 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.44538 **FUTURE DIRECTIONS**

44539 None.

44540 **SEE ALSO**44541 *mlock()*, *mlockall()*, *mmap()*, *posix\_typed\_mem\_open()*, *sysconf()*

44542 XBD &lt;sys/mman.h&gt;

44543 **CHANGE HISTORY**

44544 First released in Issue 4, Version 2.

44545 **Issue 5**

44546 Moved from X/OPEN UNIX extension to BASE.

44547 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 44548 • The DESCRIPTION is extensively reworded.
- 44549 • The SIGBUS error is no longer permitted to be generated.

44550 **Issue 6**44551 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory  
44552 Objects option.44553 The following new requirements on POSIX implementations derive from alignment with the  
44554 Single UNIX Specification:

- 44555 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple  
44556 of the page size.
- 44557 • The [EINVAL] error conditions are added.

44558 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 44559 • Semantics for typed memory objects are added to the DESCRIPTION.
- 44560 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

44561 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code  
44562 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).44563 **Issue 7**44564 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment  
44565 requirements.44566 The *munmap()* function is moved from the Memory Mapped Files option to the Base.

44567 **NAME**

44568 nan, nanf, nanl — return quiet NaN

44569 **SYNOPSIS**

```
44570 #include <math.h>
44571 double nan(const char *tagp);
44572 float nanf(const char *tagp);
44573 long double nanl(const char *tagp);
```

44574 **DESCRIPTION**

44575 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44576 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44577 volume of POSIX.1-2008 defers to the ISO C standard.

44578 The function call *nan*("*n-char-sequence*") shall be equivalent to:

```
44579 strtod("NAN(n-char-sequence)", (char **) NULL);
```

44580 The function call *nan*("") shall be equivalent to:

```
44581 strtod("NAN()", (char **) NULL)
```

44582 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be  
 44583 equivalent to:

```
44584 strtod("NAN", (char **) NULL)
```

44585 Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*()  
 44586 and *strtold*().

44587 **RETURN VALUE**

44588 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

44589 If the implementation does not support quiet NaNs, these functions shall return zero.

44590 **ERRORS**

44591 No errors are defined.

44592 **EXAMPLES**

44593 None.

44594 **APPLICATION USAGE**

44595 None.

44596 **RATIONALE**

44597 None.

44598 **FUTURE DIRECTIONS**

44599 None.

44600 **SEE ALSO**

44601 [strtod\(\)](#)

44602 XBD [<math.h>](#)

44603 **CHANGE HISTORY**

44604 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**nanosleep()**44605 **NAME**

44606 nanosleep — high resolution sleep

44607 **SYNOPSIS**

```
44608 CX #include <time.h>
44609 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

44610 **DESCRIPTION**

44611 The *nanosleep()* function shall cause the current thread to be suspended from execution until  
 44612 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the  
 44613 calling thread, and its action is to invoke a signal-catching function or to terminate the process.  
 44614 The suspension time may be longer than requested because the argument value is rounded up to  
 44615 an integer multiple of the sleep resolution or because of the scheduling of other activity by the  
 44616 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
 44617 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

44618 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.

44619 **RETURN VALUE**

44620 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall  
 44621 be zero.

44622 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a  
 44623 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the  
 44624 **timespec** structure referenced by it is updated to contain the amount of time remaining in the  
 44625 interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments may  
 44626 point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

44627 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.

44628 **ERRORS**

44629 The *nanosleep()* function shall fail if:

44630 [EINTR] The *nanosleep()* function was interrupted by a signal.

44631 [EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than  
 44632 or equal to 1 000 million.

44633 **EXAMPLES**

44634 None.

44635 **APPLICATION USAGE**

44636 None.

44637 **RATIONALE**

44638 It is common to suspend execution of a thread for an interval in order to poll the status of a non-  
 44639 interrupting function. A large number of actual needs can be met with a simple extension to  
 44640 *sleep()* that provides finer resolution.

44641 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the  
 44642 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,  
 44643 it is possible to write such a routine using no static storage and reserving no system facilities.  
 44644 Although it is possible to write a function with similar functionality to *sleep()* using the  
 44645 remainder of the *timer\_\**() functions, such a function requires the use of signals and the  
 44646 reservation of some signal number. This volume of POSIX.1-2008 requires that *nanosleep()* be  
 44647 non-intrusive of the signals function.

44648 The *nanosleep()* function shall return a value of 0 on success and `-1` on failure or if interrupted.

44649 This latter case is different from *sleep()*. This was done because the remaining time is returned  
44650 via an argument structure pointer, *rmtp*, instead of as the return value.

#### 44651 FUTURE DIRECTIONS

44652 None.

#### 44653 SEE ALSO

44654 *clock\_nanosleep()*, *sleep()*

44655 XBD <[time.h](#)>

#### 44656 CHANGE HISTORY

44657 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 44658 Issue 6

44659 The *nanosleep()* function is marked as part of the Timers option.

44660 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
44661 implementation does not support the Timers option.

44662 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO  
44663 section to include the *clock\_nanosleep()* function.

44664 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the  
44665 RATIONALE section.

#### 44666 Issue 7

44667 SD5-XBD-ERN-33 is applied.

44668 The *nanosleep()* function is moved from the Timers option to the Base.

**nearbyint()**44669 **NAME**

44670 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

44671 **SYNOPSIS**

```
44672 #include <math.h>
44673 double nearbyint(double x);
44674 float nearbyintf(float x);
44675 long double nearbyintl(long double x);
```

44676 **DESCRIPTION**

44677 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44678 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44679 volume of POSIX.1-2008 defers to the ISO C standard.

44680 These functions shall round their argument to an integer value in floating-point format, using  
 44681 the current rounding direction and without raising the inexact floating-point exception.

44682 An application wishing to check for error situations should set *errno* to zero and call  
 44683 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 44684 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 44685 zero, an error has occurred.

44686 **RETURN VALUE**

44687 Upon successful completion, these functions shall return the rounded integer value.

44688 MX If *x* is NaN, a NaN shall be returned.44689 If *x* is  $\pm 0$ ,  $\pm 0$  shall be returned.44690 If *x* is  $\pm\text{Inf}$ , *x* shall be returned.

44691 XSI If the correct value would cause overflow, a range error shall occur and *nearbyint()*, *nearbyintf()*,  
 44692 and *nearbyintl()* shall return the value of the macro  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  
 44693  $\pm\text{HUGE\_VALL}$  (with the same sign as *x*), respectively.

44694 **ERRORS**

44695 These functions shall fail if:

44696 XSI Range Error The result would cause an overflow.

44697 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 44698 then *errno* shall be set to [ERANGE]. If the integer expression  
 44699 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 44700 floating-point exception shall be raised.

44701 **EXAMPLES**

44702 None.

44703 **APPLICATION USAGE**

44704 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 44705 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

44706 **RATIONALE**

44707 None.

44708 **FUTURE DIRECTIONS**

44709 None.

44710 **SEE ALSO**44711 [fclearexcept\(\)](#), [fetestexcept\(\)](#)44712 XBD [Section 4.19](#) (on page 116), [<math.h>](#)44713 **CHANGE HISTORY**

44714 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**newlocale()**44715 **NAME**

44716 newlocale — create or modify a locale object

44717 **SYNOPSIS**

```
44718 CX #include <locale.h>
44719 locale_t newlocale(int category_mask, const char *locale,
44720 locale_t base);
```

44721 **DESCRIPTION**

44722 The *newlocale()* function shall create a new locale object or modify an existing one. If the *base*  
 44723 argument is **(locale\_t)0**, a new locale object shall be created. It is unspecified whether the locale  
 44724 object pointed to by *base* shall be modified or freed and a new locale object created.

44725 The *category\_mask* argument specifies the locale categories to be set or modified. Values for  
 44726 *category\_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants  
 44727 *LC\_CTYPE\_MASK*, *LC\_NUMERIC\_MASK*, *LC\_TIME\_MASK*, *LC\_COLLATE\_MASK*,  
 44728 *LC\_MONETARY\_MASK*, and *LC\_MESSAGES\_MASK*, or any of the other implementation-  
 44729 defined *LC\_\*\_MASK* values defined in **<locale.h>**.

44730 For each category with the corresponding bit set in *category\_mask* the data from the locale named  
 44731 by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale  
 44732 named by *locale* shall replace the existing data within the locale object. If a completely new locale  
 44733 object is created, the data for all sections not requested by *category\_mask* shall be taken from the  
 44734 default locale.

44735 The following preset values of *locale* are defined for all settings of *category\_mask*:

44736	"POSIX"	Specifies the minimal environment for C-language translation called the 44737 POSIX locale.
44738	"C"	Equivalent to "POSIX".
44739	""	Specifies an implementation-defined native environment. This corresponds to 44740 the value of the associated environment variables, <i>LC_*</i> and <i>LANG</i> ; see XBD 44741 <a href="#">Chapter 7</a> (on page 135) and <a href="#">Chapter 8</a> (on page 173).

44742 If the *base* argument is not **(locale\_t)0** and the *newlocale()* function call succeeds, the contents of  
 44743 *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before  
 44744 calling *newlocale()*. If the function call fails and the *base* argument is not **(locale\_t)0**, the contents  
 44745 of *base* shall remain valid and unchanged.

44746 The results are undefined if the *base* argument is the special locale object *LC\_GLOBAL\_LOCALE*.

44747 **RETURN VALUE**

44748 Upon successful completion, the *newlocale()* function shall return a handle which the caller may  
 44749 use on subsequent calls to *duplocale()*, *freelocale()*, and other functions taking a **locale\_t**  
 44750 argument.

44751 Upon failure, the *newlocale()* function shall return **(locale\_t)0** and set *errno* to indicate the error.

44752 **ERRORS**

44753 The *newlocale()* function shall fail if:

44754	[ENOMEM]	There is not enough memory available to create the locale object or load the 44755 locale data.
-------	----------	--

- 44756 [EINVAL] The *category\_mask* contains a bit that does not correspond to a valid category.
- 44757 [ENOENT] For any of the categories in *category\_mask*, the locale data is not available.
- 44758 The *newlocale()* function may fail if:
- 44759 [EINVAL] The *locale* argument is not a valid string pointer.

44760 **EXAMPLES**44761 **Constructing a Locale Object from Different Locales**

44762 The following example shows the construction of a locale where the *LC\_CTYPE* category data  
44763 comes from a locale *loc1* and the *LC\_TIME* category data from a locale *tok2*:

```
44764 #include <locale.h>
44765 ...
44766 locale_t loc, new_loc;
44767 /* Get the "loc1" data. */
44768 loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
44769 if (loc == (locale_t) 0)
44770     abort ();
44771 /* Get the "loc2" data. */
44772 new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
44773 if (new_loc != (locale_t) 0)
44774     /* We don't abort if this fails. In this case this
44775     simply used to unchanged locale object. */
44776     loc = new_loc;
44777 ...
```

44778 **Freeing up a Locale Object**

44779 The following example shows a code fragment to free a locale object created by *newlocale()*:

```
44780 #include <locale.h>
44781 ...
44782 /* Every locale object allocated with newlocale() should be
44783 * freed using freelocale():
44784 */
44785 locale_t loc;
44786 /* Get the locale. */
44787 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
44788 /* ... Use the locale object ... */
44789 ...
44790 /* Free the locale object resources. */
44791 freelocale (loc);
```

**newlocale()**

System Interfaces

44792 **APPLICATION USAGE**

44793 Handles for locale objects created by the *newlocale()* function should be released by a  
44794 corresponding call to *freelocale()*.

44795 The special locale object *LC\_GLOBAL\_LOCALE* must not be passed for the *base* argument, even  
44796 when returned by the *uselocale()* function.

44797 **RATIONALE**

44798 None.

44799 **FUTURE DIRECTIONS**

44800 None.

44801 **SEE ALSO**

44802 *duplocale()*, *freelocale()*, *uselocale()*

44803 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [<locale.h>](#)

44804 **CHANGE HISTORY**

44805 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

44806 **NAME**

44807 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable  
44808 floating-point number

44809 **SYNOPSIS**

```
44810 #include <math.h>
44811 double nextafter(double x, double y);
44812 float nextafterf(float x, float y);
44813 long double nextafterl(long double x, long double y);
44814 double nexttoward(double x, long double y);
44815 float nexttowardf(float x, long double y);
44816 long double nexttowardl(long double x, long double y);
```

44817 **DESCRIPTION**

44818 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
44819 conflict between the requirements described here and the ISO C standard is unintentional. This  
44820 volume of POSIX.1-2008 defers to the ISO C standard.

44821 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable  
44822 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall  
44823 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,  
44824 and *nextafterl()* functions shall return *y* if *x* equals *y*.

44825 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the  
44826 corresponding *nextafter()* functions, except that the second parameter shall have type **long**  
44827 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

44828 An application wishing to check for error situations should set *errno* to zero and call  
44829 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
44830 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
44831 zero, an error has occurred.

44832 **RETURN VALUE**

44833 Upon successful completion, these functions shall return the next representable floating-point  
44834 value following *x* in the direction of *y*.

44835 If  $x==y$ , *y* (of the type *x*) shall be returned.

44836 If *x* is finite and the correct function value would overflow, a range error shall occur and  
44837  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as  
44838 appropriate for the return type of the function.

44839 **MX** If *x* or *y* is NaN, a NaN shall be returned.

44840 If  $x!=y$  and the correct function value is subnormal, zero, or underflows, a range error shall  
44841 occur, and either the correct function value (if representable) or 0.0 shall be returned.

44842 **ERRORS**

44843 These functions shall fail if:

44844 **Range Error** The correct value overflows.

44845 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
44846 then *errno* shall be set to [ERANGE]. If the integer expression  
44847 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
44848 floating-point exception shall be raised.

**nextafter()**

44849	MX	<b>Range Error</b>	The correct value is subnormal or underflows.
44850			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
44851			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
44852			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
44853			floating-point exception shall be raised.
44854		<b>EXAMPLES</b>	
44855		None.	
44856		<b>APPLICATION USAGE</b>	
44857		On error, the expressions ( <i>math_errhandling</i> & MATH_ERRNO) and ( <i>math_errhandling</i> &	
44858		MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.	
44859		<b>RATIONALE</b>	
44860		None.	
44861		<b>FUTURE DIRECTIONS</b>	
44862		None.	
44863		<b>SEE ALSO</b>	
44864		<i>feclearexcept()</i> , <i>fetestexcept()</i>	
44865		XBD Section 4.19 (on page 116), <math.h>	
44866		<b>CHANGE HISTORY</b>	
44867		First released in Issue 4, Version 2.	
44868		<b>Issue 5</b>	
44869		Moved from X/OPEN UNIX extension to BASE.	
44870		<b>Issue 6</b>	
44871		The <i>nextafter()</i> function is no longer marked as an extension.	
44872		The <i>nextafterf()</i> , <i>nextafterl()</i> , <i>nexttoward()</i> , <i>nexttowardf()</i> , and <i>nexttowardl()</i> functions are added	
44873		for alignment with the ISO/IEC 9899:1999 standard.	
44874		The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are	
44875		revised to align with the ISO/IEC 9899:1999 standard.	
44876		IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are	
44877		marked.	

44878 **NAME**

44879 nftw — walk a file tree

44880 **SYNOPSIS**

```
44881 XSI #include <ftw.h>
44882 int nftw(const char *path, int (*fn)(const char *,
44883     const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

44884 **DESCRIPTION**

44885 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*  
 44886 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a  
 44887 bitwise-inclusive OR of zero or more of the following flags:

- 44888 **FTW\_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it  
 44889 reports files in that directory. If clear, *nftw()* shall not change the current  
 44890 working directory.
- 44891 **FTW\_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory  
 44892 itself. If clear, *nftw()* shall report any directory before reporting the files in that  
 44893 directory.
- 44894 **FTW\_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,  
 44895 *nftw()* shall report all files encountered during the walk.
- 44896 **FTW\_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

44897 If **FTW\_PHYS** is clear and **FTW\_DEPTH** is set, *nftw()* shall follow links instead of reporting  
 44898 them, but shall not report any directory that would be a descendant of itself. If **FTW\_PHYS** is  
 44899 clear and **FTW\_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not  
 44900 report the contents of any directory that would be a descendant of itself.

44901 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 44902 • The first argument is the pathname of the object.
- 44903 • The second argument is a pointer to the **stat** buffer containing information on the object,  
 44904 filled in as if *fstatat()*, *stat()*, or *lstat()* had been called to retrieve the information.
- 44905 • The third argument is an integer giving additional information. Its value is one of the  
 44906 following:
- 44907 **FTW\_D** The object is a directory.
- 44908 **FTW\_DNR** The object is a directory that cannot be read. The *fn* function shall not be  
 44909 called for any of its descendants.
- 44910 **FTW\_DP** The object is a directory and subdirectories have been visited. (This condition  
 44911 shall only occur if the **FTW\_DEPTH** flag is included in *flags*.)
- 44912 **FTW\_F** The object is a file.
- 44913 **FTW\_NS** The *stat()* function failed on the object because of lack of appropriate  
 44914 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any  
 44915 other reason is considered an error and *nftw()* shall return  $-1$ .
- 44916 **FTW\_SL** The object is a symbolic link. (This condition shall only occur if the  
 44917 **FTW\_PHYS** flag is included in *flags*.)

**nftw()**

44918 FTW\_SLN The object is a symbolic link that does not name an existing file. (This  
44919 condition shall only occur if the FTW\_PHYS flag is not included in *flags*.)

44920 • The fourth argument is a pointer to an FTW structure. The value of **base** is the offset of the  
44921 object's filename in the pathname passed as the first argument to *fn*. The value of **level**  
44922 indicates depth relative to the root of the walk, where the root level is 0.

44923 The results are unspecified if the application-supplied *fn* function does not preserve the current  
44924 working directory.

44925 The argument *fd\_limit* sets the maximum number of file descriptors that shall be used by *nftw()*  
44926 while traversing the file tree. At most one file descriptor shall be used for each directory level.

44927 The *nftw()* function need not be thread-safe.

**RETURN VALUE**

44928 The *nftw()* function shall continue until the first of the following conditions occurs:

- 44930 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that  
44931 value.
- 44932 • The *nftw()* function detects an error other than [EACCES] (see FTW\_DNR and FTW\_NS  
44933 above), in which case *nftw()* shall return -1 and set *errno* to indicate the error.
- 44934 • The tree is exhausted, in which case *nftw()* shall return 0.

**ERRORS**

44935 The *nftw()* function shall fail if:

- 44937 [EACCES] Search permission is denied for any component of *path* or read permission is  
44938 denied for *path*, or *fn* returns -1 and does not reset *errno*.
- 44939 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
44940 argument.
- 44941 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME\_MAX}.
- 44942 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 44943 [ENOTDIR] A component of *path* is not a directory.
- 44944 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
44945 programming environment for one or more files found in the file hierarchy.

44947 The *nftw()* function may fail if:

- 44948 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
44949 resolution of the *path* argument.
- 44950 [EMFILE] All file descriptors available to the process are currently open.
- 44951 [ENAMETOOLONG] The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
44952 symbolic link produced an intermediate result with a length that exceeds  
44953 {PATH\_MAX}.
- 44954 [ENFILE] Too many files are currently open in the system.

44956 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

44957 **EXAMPLES**

44958 The following program traverses the directory tree under the path named in its first command-  
 44959 line argument, or under the current directory if no argument is supplied. It displays various  
 44960 information about each file. The second command-line argument can be used to specify  
 44961 characters that control the value assigned to the *flags* argument when calling *nftw()*.

```

44962 #include <ftw.h>
44963 #include <stdio.h>
44964 #include <stdlib.h>
44965 #include <string.h>
44966 #include <stdint.h>

44967 static int
44968 display_info(const char *fpath, const struct stat *sb,
44969             int tflag, struct FTW *ftwbuf)
44970 {
44971     printf("%-3s %2d %7jd   %-40s %d %s\n",
44972          (tflag == FTW_D) ? "d" : (tflag == FTW_DNR) ? "dnr" :
44973          (tflag == FTW_DP) ? "dp" : (tflag == FTW_F) ? "f" :
44974          (tflag == FTW_NS) ? "ns" : (tflag == FTW_SL) ? "sl" :
44975          (tflag == FTW_SLN) ? "sln" : "???",
44976          ftwbuf->level, (intmax_t) sb->st_size,
44977          fpath, ftwbuf->base, fpath + ftwbuf->base);
44978     return 0; /* To tell nftw() to continue */
44979 }

44980 int
44981 main(int argc, char *argv[])
44982 {
44983     int flags = 0;

44984     if (argc > 2 && strchr(argv[2], 'd') != NULL)
44985         flags |= FTW_DEPTH;
44986     if (argc > 2 && strchr(argv[2], 'p') != NULL)
44987         flags |= FTW_PHYS;

44988     if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
44989     {
44990         perror("nftw");
44991         exit(EXIT_FAILURE);
44992     }

44993     exit(EXIT_SUCCESS);
44994 }

```

44995 **APPLICATION USAGE**

44996 The *nftw()* function may allocate dynamic storage during its operation. If *nftw()* is forcibly  
 44997 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
 44998 or an interrupt routine, *nftw()* does not have a chance to free that storage, so it remains  
 44999 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
 45000 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
 45001 invocation.

**nftw()**45002 **RATIONALE**

45003 None.

45004 **FUTURE DIRECTIONS**

45005 None.

45006 **SEE ALSO**45007 *fdopendir()*, *fstatat()*, *readdir()*

45008 XBD &lt;ftw.h&gt;

45009 **CHANGE HISTORY**

45010 First released in Issue 4, Version 2.

45011 **Issue 5**

45012 Moved from X/OPEN UNIX extension to BASE.

45013 In the DESCRIPTION, the definition of the *depth* argument is clarified.45014 **Issue 6**

45015 The Open Group Base Resolution bwg97-003 is applied.

45016 The ERRORS section is updated as follows:

- 45017 • The wording of the mandatory [ELOOP] error condition is updated.
- 45018 • A second optional [ELOOP] error condition is added.
- 45019 • The [EOVERFLOW] mandatory error condition is added.

45020 Text is added to the DESCRIPTION to say that the *nftw()* function need not be reentrant and that  
 45021 the results are unspecified if the application-supplied *fn* function does not preserve the current  
 45022 working directory.

45023 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument  
 45024 *depth* to *fd\_limit* throughout and changing “to a maximum of 5 levels deep” to “using a  
 45025 maximum of 5 file descriptors” in the EXAMPLES section.

45026 **Issue 7**

45027 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

45028 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

45029 SD5-XBD-ERN-61 is applied.

45030 APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

45031 **NAME**

45032 nice — change the nice value of a process

45033 **SYNOPSIS**

```
45034 XSI #include <unistd.h>
45035 int nice(int incr);
```

45036 **DESCRIPTION**

45037 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A nice  
 45038 value of a process is a non-negative number for which a more positive value shall result in less  
 45039 favorable scheduling.

45040 A maximum nice value of  $2 * \{NZERO\} - 1$  and a minimum nice value of 0 shall be imposed by the  
 45041 system. Requests for values above or below these limits shall result in the nice value being set to  
 45042 the corresponding limit. Only a process with appropriate privileges can lower the nice value.

45043 PS | TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy  
 45044 SCHED\_FIFO or SCHED\_RR. The effect on processes or threads with other scheduling policies  
 45045 is implementation-defined.

45046 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the  
 45047 nice value shall affect all system scope threads in the process.

45048 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for  
 45049 error situations should set *errno* to 0, then call *nice()*, and if it returns  $-1$ , check to see whether  
 45050 *errno* is non-zero.

45051 **RETURN VALUE**

45052 Upon successful completion, *nice()* shall return the new nice value  $-\{NZERO\}$ . Otherwise,  $-1$   
 45053 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to  
 45054 indicate the error.

45055 **ERRORS**

45056 The *nice()* function shall fail if:

45057 [EPERM] The *incr* argument is negative and the calling process does not have  
 45058 appropriate privileges.

45059 **EXAMPLES**45060 **Changing the Nice Value**

45061 The following example adds the value of the *incr* argument,  $-20$ , to the nice value of the calling  
 45062 process.

```
45063 #include <unistd.h>
45064 ...
45065 int incr = -20;
45066 int ret;
45067 ret = nice(incr);
```

45068 **APPLICATION USAGE**

45069 None.

**nice()**45070 **RATIONALE**

45071 None.

45072 **FUTURE DIRECTIONS**

45073 None.

45074 **SEE ALSO**45075 *exec*, *getpriority()*45076 XBD [<limits.h>](#), [<unistd.h>](#)45077 **CHANGE HISTORY**

45078 First released in Issue 1. Derived from Issue 1 of the SVID.

45079 **Issue 5**45080 A statement is added to the description indicating the effects of this function on the different  
45081 scheduling policies and multi-threaded processes.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

45082 **NAME**

45083 nl\_langinfo, nl\_langinfo\_l — language information

45084 **SYNOPSIS**

```
45085 #include <langinfo.h>
45086 char *nl_langinfo(nl_item item);
45087 char *nl_langinfo_l(nl_item item, locale_t locale);
```

45088 **DESCRIPTION**

45089 The *nl\_langinfo()* and *nl\_langinfo\_l()* functions shall return a pointer to a string containing  
 45090 information relevant to the particular language or cultural area defined in the locale of the  
 45091 process, or in the locale represented by *locale*, respectively (see <langinfo.h>). The manifest  
 45092 constant names and values of *item* are defined in <langinfo.h>. For example:

```
45093 nl_langinfo(ABDAY_1)
```

45094 would return a pointer to the string "Dom" if the identified language was Portuguese, and  
 45095 "Sun" if the identified language was English.

```
45096 nl_langinfo_l(ABDAY_1, loc)
```

45097 would return a pointer to the string "Dom" if the identified language of the locale represented by  
 45098 *loc* was Portuguese, and "Sun" if the identified language of the locale represented by *loc* was  
 45099 English.

45100 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to  
 45101 the category *LC\_ALL*, may overwrite the array pointed to by the return value. Calls to *uselocale()*  
 45102 which change the category corresponding to the category of *item* may overwrite the array  
 45103 pointed to by the return value.

45104 The *nl\_langinfo()* function need not be thread-safe.

45105 **RETURN VALUE**

45106 In a locale where *langinfo* data is not defined, these functions shall return a pointer to the  
 45107 corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to  
 45108 an empty string if *item* contains an invalid setting.

45109 This pointer may point to static data that may be overwritten on the next call to either function.

45110 **ERRORS**

45111 The *nl\_langinfo\_l()* function may fail if:

45112 [EINVAL] *locale* is not a valid locale object handle.

45113 **EXAMPLES**45114 **Getting Date and Time Formatting Information**

45115 The following example returns a pointer to a string containing date and time formatting  
 45116 information, as defined in the *LC\_TIME* category of the current locale.

```
45117 #include <time.h>
45118 #include <langinfo.h>
45119 ...
45120 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
45121 ...
```

**nl\_langinfo()**

System Interfaces

45122 **APPLICATION USAGE**

45123 The array pointed to by the return value should not be modified by the program, but may be  
45124 modified by further calls to these functions.

45125 **RATIONALE**

45126 None.

45127 **FUTURE DIRECTIONS**

45128 None.

45129 **SEE ALSO**

45130 [setlocale\(\)](#), [uselocale\(\)](#)

45131 XBD [Chapter 7](#) (on page 135), [<langinfo.h>](#), [<locale.h>](#), [<nl\\_types.h>](#)

45132 **CHANGE HISTORY**

45133 First released in Issue 2.

45134 **Issue 5**

45135 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

45136 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

45137 **Issue 7**

45138 Austin Group Interpretation 1003.1-2001 #156 is applied.

45139 The `nl_langinfo()` function is moved from the XSI option to the Base.

45140 The `nl_langinfo_l()` function is added from The Open Group Technical Standard, 2006, Extended  
45141 API Set Part 4.

45142 **NAME**

45143 nrnd48 — generate uniformly distributed pseudo-random non-negative long integers

45144 **SYNOPSIS**45145 XSI 

```
#include <stdlib.h>
```

  
45146 

```
long nrnd48(unsigned short xsubi[3]);
```

45147 **DESCRIPTION**45148 Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ntohl()**45149 **NAME**

45150           ntohl, ntohs — convert values between host and network byte order

45151 **SYNOPSIS**

```
45152       #include <arpa/inet.h>
45153       uint32_t ntohl(uint32_t netlong);
45154       uint16_t ntohs(uint16_t netshort);
```

45155 **DESCRIPTION**45156       Refer to *htonl()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

45157 **NAME**

45158 open, openat — open file relative to directory file descriptor

45159 **SYNOPSIS**45160 OH `#include <sys/stat.h>`45161 `#include <fcntl.h>`45162 `int open(const char *path, int oflag, ...);`45163 `int openat(int fd, const char *path, int oflag, ...);`45164 **DESCRIPTION**

45165 The `open()` function shall establish the connection between a file and a file descriptor. It shall  
 45166 create an open file description that refers to a file and a file descriptor that refers to that open file  
 45167 description. The file descriptor is used by other I/O functions to refer to that file. The `path`  
 45168 argument points to a pathname naming the file.

45169 The `open()` function shall return a file descriptor for the named file that is the lowest file  
 45170 descriptor not currently open for that process. The open file description is new, and therefore the  
 45171 file descriptor shall not share it with any other process in the system. The `FD_CLOEXEC` file  
 45172 descriptor flag associated with the new file descriptor shall be cleared unless the `O_CLOEXEC`  
 45173 flag is set in `oflag`.

45174 The file offset used to mark the current position within the file shall be set to the beginning of  
 45175 the file.

45176 The file status flags and file access modes of the open file description shall be set according to  
 45177 the value of `oflag`.

45178 Values for `oflag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 45179 in `<fcntl.h>`. Applications shall specify exactly one of the first five values (file access modes)  
 45180 below in the value of `oflag`:

45181 `O_EXEC` Open for execute only (non-directory files). The result is unspecified if  
 45182 this flag is applied to a directory.

45183 `O_RDONLY` Open for reading only.

45184 `O_RDWR` Open for reading and writing. The result is undefined if this flag is  
 45185 applied to a FIFO.

45186 `O_SEARCH` Open directory for search only. The result is unspecified if this flag is  
 45187 applied to a non-directory file.

45188 `O_WRONLY` Open for writing only.

45189 Any combination of the following may be used:

45190 `O_APPEND` If set, the file offset shall be set to the end of the file prior to each write.

45191 `O_CLOEXEC` If set, the `FD_CLOEXEC` flag for the new file descriptor shall be set.

45192 `O_CREAT` If the file exists, this flag has no effect except as noted under `O_EXCL`  
 45193 below. Otherwise, the file shall be created; the user ID of the file shall be  
 45194 set to the effective user ID of the process; the group ID of the file shall be  
 45195 set to the group ID of the file's parent directory or to the effective group  
 45196 ID of the process; and the access permission bits (see `<sys/stat.h>`) of the  
 45197 file mode shall be set to the value of the argument following the `oflag`  
 45198 argument taken as type `mode_t` modified as follows: a bitwise AND is  
 45199 performed on the file-mode bits and the corresponding bits in the  
 45200 complement of the process' file mode creation mask. Thus, all bits in the

## open()

45201		file mode whose corresponding bit in the file mode creation mask is set
45202		are cleared. When bits other than the file permission bits are set, the effect
45203		is unspecified. The argument following the <i>oflag</i> argument does not affect
45204		whether the file is open for reading, writing, or for both. Implementations
45205		shall provide a way to initialize the file's group ID to the group ID of the
45206		parent directory. Implementations may, but need not, provide an
45207		implementation-defined way to initialize the file's group ID to the
45208		effective group ID of the calling process.
45209	O_DIRECTORY	If <i>path</i> does not name a directory, fail and set <i>errno</i> to [ENOTDIR].
45210	SIO O_DSYNC	Write I/O operations on the file descriptor shall complete as defined by
45211		synchronized I/O data integrity completion.
45212	O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The
45213		check for the existence of the file and the creation of the file if it does not
45214		exist shall be atomic with respect to other threads executing <i>open()</i>
45215		naming the same filename in the same directory with O_EXCL and
45216		O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a
45217		symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the
45218		contents of the symbolic link. If O_EXCL is set and O_CREAT is not set,
45219		the result is undefined.
45220	O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the
45221		terminal device to become the controlling terminal for the process. If <i>path</i>
45222		does not identify a terminal device, O_NOCTTY shall be ignored.
45223	O_NOFOLLOW	If <i>path</i> names a symbolic link, fail and set <i>errno</i> to [ELOOP].
45224	O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set:
45225		• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return
45226		without delay. An <i>open()</i> for writing-only shall return an error if no
45227		process currently has the file open for reading.
45228		• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the
45229		calling thread until a thread opens the file for writing. An <i>open()</i> for
45230		writing-only shall block the calling thread until a thread opens the
45231		file for reading.
45232		When opening a block special or character special file that supports non-
45233		blocking opens:
45234		• If O_NONBLOCK is set, the <i>open()</i> function shall return without
45235		blocking for the device to be ready or available. Subsequent
45236		behavior of the device is device-specific.
45237		• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling
45238		thread until the device is ready or available before returning.
45239		Otherwise, the behavior of O_NONBLOCK is unspecified.
45240	SIO O_RSYNC	Read I/O operations on the file descriptor shall complete at the same
45241		level of integrity as specified by the O_DSYNC and O_SYNC flags. If both
45242		O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file
45243		descriptor shall complete as defined by synchronized I/O data integrity
45244		completion. If both O_SYNC and O_RSYNC are set in flags, all I/O
45245		operations on the file descriptor shall complete as defined by

45246			synchronized I/O file integrity completion.
45247	XSI   SIO	O_SYNC	Write I/O operations on the file descriptor shall complete as defined by
45248			synchronized I/O file integrity completion.
45249	XSI		The O_SYNC flag shall be supported for regular files, even if the
45250			Synchronized Input and Output option is not supported.
45251		O_TRUNC	If the file exists and is a regular file, and the file is successfully opened
45252			O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode
45253			and owner shall be unchanged. It shall have no effect on FIFO special files
45254			or terminal device files. Its effect on other file types is implementation-
45255			defined. The result of using O_TRUNC without either O_RDWR or
45256			O_WRONLY is undefined.
45257		O_TTY_INIT	If <i>path</i> identifies a terminal device other than a pseudo-terminal, the
45258			device is not already open in any process, and either O_TTY_INIT is set in
45259			<i>oflag</i> or O_TTY_INIT has the value zero, <i>open()</i> shall set any non-standard
45260			<b>termios</b> structure terminal parameters to a state that provides conforming
45261			behavior; see XBD Section 11.2 (on page 205). It is unspecified whether
45262			O_TTY_INIT has any effect if the device is already open in any process. If
45263			<i>path</i> identifies the slave side of a pseudo-terminal that is not already open
45264			in any process, <i>open()</i> shall set any non-standard <b>termios</b> structure
45265			terminal parameters to a state that provides conforming behavior,
45266			regardless of whether O_TTY_INIT is set. If <i>path</i> does not identify a
45267			terminal device, O_TTY_INIT shall be ignored.
45268			If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall
45269			mark for update the last data access, last data modification, and last file status change
45270			timestamps of the file and the last data modification and last file status change timestamps of
45271			the parent directory.
45272			If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall
45273			mark for update the last data modification and last file status change timestamps of the file.
45274	SIO		If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.
45275	OB   XSR		If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with
45276			either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS
45277			devices and shall have no effect on them. The value O_NONBLOCK affects the operation of
45278			STREAMS drivers and certain functions applied to file descriptors associated with STREAMS
45279			files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.
45280			The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal
45281			device since system boot or since the device was closed by the process that last had it open. The
45282	XSI		application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If <i>path</i>
45283			names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks
45284			the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before
45285			opening the slave side.
45286			The largest value that can be represented correctly in an object of type <b>off_t</b> shall be established
45287			as the offset maximum in the open file description.
45288			The <i>openat()</i> function shall be equivalent to the <i>open()</i> function except in the case where <i>path</i>
45289			specifies a relative path. In this case the file to be opened is determined relative to the directory
45290			associated with the file descriptor <i>fd</i> instead of the current working directory. If the file

**open()**

45291 descriptor was opened without `O_SEARCH`, the function shall check whether directory searches  
 45292 are permitted using the current permissions of the directory underlying the file descriptor. If the  
 45293 file descriptor was opened with `O_SEARCH`, the function shall not perform the check.

45294 The *oflag* parameter and the optional fourth parameter correspond exactly to the parameters of  
 45295 *open()*.

45296 If *openat()* is passed the special value `AT_FDCWD` in the *fd* parameter, the current working  
 45297 directory is used and the behavior shall be identical to a call to *open()*.

**RETURN VALUE**

45298 Upon successful completion, these functions shall open the file and return a non-negative  
 45299 integer representing the lowest numbered unused file descriptor. Otherwise, these functions  
 45300 shall return `-1` and set *errno* to indicate the error. If `-` is returned, no files shall be created or  
 45301 modified.  
 45302

**ERRORS**

45303 These functions shall fail if:

45304 [EACCES] Search permission is denied on a component of the path prefix, or the file  
 45305 exists and the permissions specified by *oflag* are denied, or the file does not  
 45306 exist and write permission is denied for the parent directory of the file to be  
 45307 created, or `O_TRUNC` is specified and write permission is denied.  
 45308

45309 [EEXIST] `O_CREAT` and `O_EXCL` are set, and the named file exists.

45310 [EINTR] A signal was caught during *open()*.

45311 SIO [EINVAL] The implementation does not support synchronized I/O for this file.

45312 OB XSR [EIO] The *path* argument names a STREAMS file and a hangup or error occurred  
 45313 during the *open()*.

45314 [EISDIR] The named file is a directory and *oflag* includes `O_WRONLY` or `O_RDWR`.

45315 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 45316 argument, or `O_NOFOLLOW` was specified and the *path* argument names a  
 45317 symbolic link.

45318 [EMFILE] All file descriptors available to the process are currently open.

45319 [ENAMETOOLONG]

45320 The length of a component of a pathname is longer than `{NAME_MAX}`.

45321 [ENFILE] The maximum allowable number of files is currently open in the system.

45322 [ENOENT] `O_CREAT` is not set and the named file does not exist; or `O_CREAT` is set and  
 45323 either the path prefix does not exist or the *path* argument points to an empty  
 45324 string.

45325 OB XSR [ENOSR] The *path* argument names a STREAMS-based file and the system is unable to  
 45326 allocate a STREAM.

45327 [ENOSPC] The directory or file system that would contain the new file cannot be  
 45328 expanded, the file does not exist, and `O_CREAT` is specified.

45329 [ENOTDIR] A component of the path prefix is not a directory; or `O_CREAT` and `O_EXCL`  
 45330 are not specified, the *path* argument contains at least one non-`<slash>`  
 45331 character and ends with one or more trailing `<slash>` characters, and the last  
 45332 pathname component names an existing file that is neither a directory nor a  
 45333 symbolic link to a directory; or `O_DIRECTORY` was specified and the *path*

45334		argument does not name a directory.
45335	[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.
45336		
45337	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
45338		
45339	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
45340		
45341	[EROFS]	The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i> argument.
45342		
45343		
45344		The <i>openat()</i> function shall fail if:
45345	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
45346		
45347	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
45348		
45349		These functions may fail if:
45350	XSI [EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.
45351		
45352	[EINVAL]	The value of the <i>oflag</i> argument is not valid.
45353	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
45354		
45355	[ENAMETOOLONG]	
45356		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
45357		
45358		
45359	OB XSR [ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.
45360		
45361	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is O_WRONLY or O_RDWR.
45362		
45363		The <i>openat()</i> function may fail if:
45364	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
45365		

45366 **EXAMPLES**45367 **Opening a File for Writing by the Owner**

45368 The following example opens the file `/tmp/file`, either by creating it (if it does not already exist),  
 45369 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,  
 45370 the access permission bits in the file mode of the file are set to permit reading and writing by the  
 45371 owner, and to permit reading only by group members and others.

45372 If the call to *open()* is successful, the file is opened for writing.

```
45373 #include <fcntl.h>
45374 ...
```

**open()**

```

45375     int fd;
45376     mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
45377     char *filename = "/tmp/file";
45378     ...
45379     fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
45380     ...

```

**Opening a File Using an Existence Check**

The following example uses the *open()* function to try to create the **LOCKFILE** file and open it for writing. Since the *open()* function specifies the **O\_EXCL** flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```

45386     #include <fcntl.h>
45387     #include <stdio.h>
45388     #include <stdlib.h>
45389
45390     #define LOCKFILE "/etc/ptmp"
45391     ...
45392     int pfd; /* Integer for file descriptor returned by open() call. */
45393     ...
45394     if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
45395                   S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
45396     {
45397         fprintf(stderr, "Cannot open %etc/ptmp. Try again later.\n");
45398         exit(1);
45399     }
45400     ...

```

**Opening a File for Writing**

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```

45403     #include <fcntl.h>
45404     #include <stdio.h>
45405     #include <stdlib.h>
45406
45407     #define LOCKFILE "/etc/ptmp"
45408     ...
45409     int pfd;
45410     char filename[PATH_MAX+1];
45411     ...
45412     if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
45413                   S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
45414     {
45415         perror("Cannot open output file\n"); exit(1);
45416     }
45417     ...

```

## 45417 APPLICATION USAGE

45418 POSIX.1-2008 does not require that terminal parameters be automatically set to any state on first  
 45419 open, nor that they be reset after the last close. It is possible for a non-conforming application to  
 45420 leave a terminal device in a state where the next process to use that device finds it in a non-  
 45421 conforming state, but has no way of determining this. To ensure that the device is set to a  
 45422 conforming initial state, applications which perform a first open of a terminal (other than a  
 45423 pseudo-terminal) should do so using the O\_TTY\_INIT flag to set the parameters associated with  
 45424 the terminal to a conforming state.

## 45425 RATIONALE

45426 Except as specified in this volume of POSIX.1-2008, the flags allowed in *oflag* are not mutually-  
 45427 exclusive and any number of them may be used simultaneously.

45428 Some implementations permit opening FIFOs with O\_RDWR. Since FIFOs could be  
 45429 implemented in other ways, and since two file descriptors can be used to the same effect, this  
 45430 possibility is left as undefined.

45431 See *getgroups()* about the group of a newly created file.

45432 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is  
 45433 redundant and included only for historical reasons.

45434 The use of the O\_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be  
 45435 permissible without unexpected side-effects (for example, *creat()* on a FIFO must not remove  
 45436 data). Since terminal special files might have type-ahead data stored in the buffer, O\_TRUNC  
 45437 should not affect their content, particularly if a program that normally opens a regular file  
 45438 should open the current controlling terminal instead. Other file types, particularly  
 45439 implementation-defined ones, are left implementation-defined.

45440 POSIX.1-2008 permits [EACCES] to be returned for conditions other than those explicitly listed.

45441 The O\_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a  
 45442 controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2008 does  
 45443 not specify how a controlling terminal is acquired, but it allows an implementation to provide  
 45444 this on *open()* if the O\_NOCTTY flag is not set and other conditions specified in XBD Chapter 11  
 45445 (on page 199) are met.

45446 In historical implementations the value of O\_RDONLY is zero. Because of that, it is not possible  
 45447 to detect the presence of O\_RDONLY and another option. Future implementations should  
 45448 encode O\_RDONLY and O\_WRONLY as bit flags so that:

45449 `O_RDONLY | O_WRONLY == O_RDWR`

45450 O\_EXEC and O\_SEARCH are specified as two of the five file access modes. Since O\_EXEC does  
 45451 not apply to directories, and O\_SEARCH only applies to directories, their values need not be  
 45452 distinct. Since O\_RDONLY has historically had the value zero, implementations are not able to  
 45453 distinguish between O\_SEARCH and O\_SEARCH | O\_RDONLY, and similarly for O\_EXEC.

45454 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,  
 45455 the *open()* function, when called with O\_CREAT and O\_EXCL, is required to fail with [EEXIST]  
 45456 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This  
 45457 behavior is required so that privileged applications can create a new file in a known location  
 45458 without the possibility that a symbolic link might cause the file to be created in a different  
 45459 location.

45460 For example, a privileged application that must create a file with a predictable name in a user-  
 45461 writable directory, such as the user's home directory, could be compromised if the user creates a  
 45462 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can

**open()**

45463 influence the contents of a file, the user could compromise the system by creating a new system  
 45464 configuration or spool file that would then be interpreted by the system. The test for a symbolic  
 45465 link which refers to a nonexisting file must be atomic with the creation of a new file.

45466 In addition, the *open()* function refuses to open non-directories if the `O_DIRECTORY` flag is set.  
 45467 This avoids race conditions whereby a user might compromise the system by substituting a hard  
 45468 link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where  
 45469 opening a file even for read access might have undesirable side-effects.

45470 In addition, the *open()* function does not follow symbolic links if the `O_NOFOLLOW` flag is set.  
 45471 This avoids race conditions whereby a user might compromise the system by substituting a  
 45472 symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where  
 45473 opening a file even for read access might have undesirable side-effects.

45474 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
 45475 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
 45476 that implementations provide a way to have the group ID be set to the group ID of the  
 45477 containing directory, but did not prohibit implementations also supporting a way to set the  
 45478 group ID to the effective group ID of the creating process. Conforming applications should not  
 45479 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
 45480 group ID after the file is created, or determine under what conditions the implementation will  
 45481 set the desired group ID.

45482 The purpose of the *openat()* function is to enable opening files in directories other than the  
 45483 current working directory without exposure to race conditions. Any part of the path of a file  
 45484 could be changed in parallel to a call to *open()*, resulting in unspecified behavior. By opening a  
 45485 file descriptor for the target directory and using the *openat()* function it can be guaranteed that  
 45486 the opened file is located relative to the desired directory. Some implementations use the  
 45487 *openat()* function for other purposes as well. In some cases, if the *oflag* parameter has the  
 45488 `O_XATTR` bit set, the returned file descriptor provides access to extended attributes. This  
 45489 functionality is not standardized here.

**FUTURE DIRECTIONS**

45490 None.

**SEE ALSO**

45493 *chmod()*, *close()*, *creat()*, *dirfd()*, *dup()*, *exec*, *fcntl()*, *fdopendir()*, *link()*, *lseek()*, *mkdtemp()*,  
 45494 *mknod()*, *read()*, *symlink()*, *umask()*, *unlockpt()*, *write()*

45495 XBD Chapter 41 (on page 199), `<fcntl.h>`, `<sys/stat.h>`, `<sys/types.h>`

**CHANGE HISTORY**

45497 First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

45499 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 45500 Threads Extension.

45501 Large File Summit extensions are added.

**Issue 6**

45503 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

45504 The following new requirements on POSIX implementations derive from alignment with the  
 45505 Single UNIX Specification:

- 45506 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 45507 required for conforming implementations of previous POSIX specifications, it was not  
 45508 required for UNIX applications.
- 45509 • In the DESCRIPTION, `O_CREAT` is amended to state that the group ID of the file is set to  
 45510 the group ID of the file's parent directory or to the effective group ID of the process. This is  
 45511 a FIPS requirement.
- 45512 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
 45513 file description. This change is to support large files.
- 45514 • In the ERRORS section, the `[Eoverflow]` condition is added. This change is to support  
 45515 large files.
- 45516 • The `[ENXIO]` mandatory error condition is added.
- 45517 • The `[EINVAL]`, `[ENAMETOOLONG]`, and `[ETXTBSY]` optional error conditions are added.
- 45518 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI  
 45519 STREAMS Option Group are marked.
- 45520 The following changes were made to align with the IEEE P1003.1a draft standard:
- 45521 • An explanation is added of the effect of the `O_CREAT` and `O_EXCL` flags when the path  
 45522 refers to a symbolic link.
- 45523 • The `[ELOOP]` optional error condition is added.
- 45524 The normative text is updated to avoid use of the term "must" for application requirements.
- 45525 The DESCRIPTION of `O_EXCL` is updated in response to IEEE PASC Interpretation 1003.1c #48.
- 45526 **Issue 7**
- 45527 Austin Group Interpretations 1003.1-2001 #113 and #143 are applied.
- 45528 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the `O_TTY_INIT` flag.
- 45529 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
 45530 `FD_CLOEXEC` flag atomically at `open()`, and adding the `F_DUPFD_CLOEXEC` flag.
- 45531 SD5-XBD-ERN-4 is applied, changing the definition of the `[EMFILE]` error.
- 45532 This page is revised and the `openat()` function is added from The Open Group Technical  
 45533 Standard, 2006, Extended API Set Part 2.
- 45534 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 45535 Changes are made related to support for finegrained timestamps.
- 45536 Changes are made to allow a directory to be opened for searching.

**open\_memstream()**45537 **NAME**45538 `open_memstream, open_wmemstream` — open a dynamic memory buffer stream45539 **SYNOPSIS**

```

45540 CX #include <stdio.h>
45541 FILE *open_memstream(char **bufp, size_t *sizep);
45542 #include <wchar.h>
45543 FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);

```

45544 **DESCRIPTION**

45545 The `open_memstream()` and `open_wmemstream()` functions shall create an I/O stream associated  
 45546 with a dynamically allocated memory buffer. The stream shall be opened for writing and shall  
 45547 be seekable.

45548 The stream associated with a call to `open_memstream()` shall be byte-oriented.

45549 The stream associated with a call to `open_wmemstream()` shall be wide-oriented.

45550 The stream shall maintain a current position in the allocated buffer and a current buffer length.  
 45551 The position shall be initially set to zero (the start of the buffer). Each write to the stream shall  
 45552 start at the current position and move this position by the number of successfully written bytes  
 45553 for `open_memstream()` or the number of successfully written wide characters for  
 45554 `open_wmemstream()`. The length shall be initially set to zero. If a write moves the position to a  
 45555 value larger than the current length, the current length shall be set to this position. In this case a  
 45556 null character for `open_memstream()` or a null wide character for `open_wmemstream()` shall be  
 45557 appended to the current buffer. For both functions the terminating null is not included in the  
 45558 calculation of the buffer length.

45559 After a successful `fflush()` or `fclose()`, the pointer referenced by `bufp` shall contain the address of  
 45560 the buffer, and the variable pointed to by `sizep` shall contain the smaller of the current buffer  
 45561 length and the number of bytes for `open_memstream()`, or the number of wide characters for  
 45562 `open_wmemstream()`, between the beginning of the buffer and the current file position indicator.

45563 After a successful `fflush()` the pointer referenced by `bufp` and the variable referenced by `sizep`  
 45564 remain valid only until the next write operation on the stream or a call to `fclose()`.

45565 **RETURN VALUE**

45566 Upon successful completion, these functions shall return a pointer to the object controlling the  
 45567 stream. Otherwise, a null pointer shall be returned, and `errno` shall be set to indicate the error.

45568 **ERRORS**

45569 These functions may fail if:

- |       |          |  |
|-------|----------|--|
| 45570 | [EINVAL] | <code>bufp</code> or <code>sizep</code> are NULL.              |
| 45571 | [EMFILE] | {FOPEN_MAX} streams are currently open in the calling process. |
| 45572 | [ENOMEM] | Memory for the stream or the buffer could not be allocated.    |

45573 **EXAMPLES**

```

45574     #include <stdio.h>
45575     #include <stdlib.h>

45576     int
45577     main (void)
45578     {
45579         FILE *stream;
45580         char *buf;
45581         size_t len;
45582         off_t eob;

45583         stream = open_memstream (&buf, &len);
45584         if (stream == NULL)
45585             /* handle error */ ;
45586         fprintf (stream, "hello my world");
45587         fflush (stream);
45588         printf ("buf=%s, len=%zu\n", buf, len);
45589         eob = ftello(stream);
45590         fseeko (stream, 0, SEEK_SET);
45591         fprintf (stream, "good-bye");
45592         fseeko (stream, eob, SEEK_SET);
45593         fclose (stream);
45594         printf ("buf=%s, len=%zu\n", buf, len);
45595         free (buf);
45596         return 0;
45597     }

```

45598 This program produces the following output:

```

45599     buf=hello my world, len=14
45600     buf=good-bye world, len=14

```

45601 **APPLICATION USAGE**

45602 The buffer created by these functions should be freed by the application after closing the stream,  
45603 by means of a call to *free()*.

45604 **RATIONALE**

45605 These functions are similar to *fmemopen()* except that the memory is always allocated  
45606 dynamically by the function, and the stream is opened only for output.

45607 **FUTURE DIRECTIONS**

45608 None.

45609 **SEE ALSO**

45610 *fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *free()*, *freopen()*

45611 XBD <stdio.h>, <wchar.h>

45612 **CHANGE HISTORY**

45613 First released in Issue 7.

**openat()**45614 **NAME**

45615            openat — open file relative to directory file descriptor

45616 **SYNOPSIS**

45617            #include &lt;fcntl.h&gt;

45618            int openat(int *fd*, const char \**path*, int *oflag*, ...);45619 **DESCRIPTION**45620            Refer to *open()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***opendir()**45621 **NAME**

45622       opendir — open directory associated with file descriptor

45623 **SYNOPSIS**

45624       #include &lt;dirent.h&gt;

45625       DIR \*opendir(const char \*dirname);

45626 **DESCRIPTION**45627       Refer to *fdopendir()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**openlog()***System Interfaces*45628 **NAME**

45629           openlog — open a connection to the logging facility

45630 **SYNOPSIS**

45631 XSI       #include &lt;syslog.h&gt;

45632       void openlog(const char \*ident, int logopt, int facility);

45633 **DESCRIPTION**45634       Refer to *closelog()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***optarg**45635 **NAME**

45636           optarg, opterr, optind, optopt — options parsing variables

45637 **SYNOPSIS**

45638           #include &lt;unistd.h&gt;

45639           extern char \*optarg;

45640           extern int opterr, optind, optopt;

45641 **DESCRIPTION**45642           Refer to *getopt()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pathconf()***System Interfaces*45643 **NAME**45644 `pathconf` — get configurable pathname variables45645 **SYNOPSIS**45646 `#include <unistd.h>`45647 `long pathconf(const char *path, int name);`45648 **DESCRIPTION**45649 Refer to *fpathconf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

45650 **NAME**

45651 pause — suspend the thread until a signal is received

45652 **SYNOPSIS**

45653 #include &lt;unistd.h&gt;

45654 int pause(void);

45655 **DESCRIPTION**45656 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is  
45657 either to execute a signal-catching function or to terminate the process.45658 If the action is to terminate the process, *pause()* shall not return.45659 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching  
45660 function returns.45661 **RETURN VALUE**45662 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no  
45663 successful completion return value. A value of  $-1$  shall be returned and *errno* set to indicate the  
45664 error.45665 **ERRORS**45666 The *pause()* function shall fail if:45667 [EINTR] A signal is caught by the calling process and control is returned from the  
45668 signal-catching function.45669 **EXAMPLES**

45670 None.

45671 **APPLICATION USAGE**45672 Many common uses of *pause()* have timing windows. The scenario involves checking a  
45673 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal  
45674 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The  
45675 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.45676 **RATIONALE**

45677 None.

45678 **FUTURE DIRECTIONS**

45679 None.

45680 **SEE ALSO**45681 *sigsuspend()*

45682 XBD &lt;unistd.h&gt;

45683 **CHANGE HISTORY**

45684 First released in Issue 1. Derived from Issue 1 of the SVID.

45685 **Issue 5**

45686 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

45687 **Issue 6**

45688 The APPLICATION USAGE section is added.

**pclose()**45689 **NAME**45690 `pclose` — close a pipe stream to or from a process45691 **SYNOPSIS**

```
45692 CX #include <stdio.h>
45693 int pclose(FILE *stream);
```

45694 **DESCRIPTION**

45695 The `pclose()` function shall close a stream that was opened by `popen()`, wait for the command to  
 45696 terminate, and return the termination status of the process that was running the command  
 45697 language interpreter. However, if a call caused the termination status to be unavailable to  
 45698 `pclose()`, then `pclose()` shall return `-1` with `errno` set to `[ECHILD]` to report this situation. This can  
 45699 happen if the application calls one of the following functions:

- 45700 • `wait()`
- 45701 • `waitpid()` with a `pid` argument less than or equal to 0 or equal to the process ID of the  
 45702 command line interpreter
- 45703 • Any other function not defined in this volume of POSIX.1-2008 that could do one of the  
 45704 above

45705 In any case, `pclose()` shall not return before the child process created by `popen()` has terminated.

45706 If the command language interpreter cannot be executed, the child termination status returned  
 45707 by `pclose()` shall be as if the command language interpreter terminated using `exit(127)` or  
 45708 `_exit(127)`.

45709 The `pclose()` function shall not affect the termination status of any child of the calling process  
 45710 other than the one created by `popen()` for the associated stream.

45711 If the argument `stream` to `pclose()` is not a pointer to a stream created by `popen()`, the result of  
 45712 `pclose()` is undefined.

45713 **RETURN VALUE**

45714 Upon successful return, `pclose()` shall return the termination status of the command language  
 45715 interpreter. Otherwise, `pclose()` shall return `-1` and set `errno` to indicate the error.

45716 **ERRORS**

45717 The `pclose()` function shall fail if:

- 45718 `[ECHILD]` The status of the child process could not be obtained, as described above.

45719 **EXAMPLES**

45720 None.

45721 **APPLICATION USAGE**

45722 None.

45723 **RATIONALE**

45724 There is a requirement that `pclose()` not return before the child process terminates. This is  
 45725 intended to disallow implementations that return `[EINTR]` if a signal is received while waiting.  
 45726 If `pclose()` returned before the child terminated, there would be no way for the application to  
 45727 discover which child used to be associated with the stream, and it could not do the cleanup  
 45728 itself.

45729 If the stream pointed to by `stream` was not created by `popen()`, historical implementations of  
 45730 `pclose()` return `-1` without setting `errno`. To avoid requiring `pclose()` to set `errno` in this case,  
 45731 POSIX.1-2008 makes the behavior unspecified. An application should not use `pclose()` to close

45732 any stream that was not created by *popen()*.

45733 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,  
45734 and SIGHUP while waiting for the child process to terminate. Since this behavior is not  
45735 described for the *pclose()* function in POSIX.1-2008, such implementations are not conforming.  
45736 Also, some historical implementations return [EINTR] if a signal is received, even though the  
45737 child process has not terminated. Such implementations are also considered non-conforming.

45738 Consider, for example, an application that uses:

```
45739 popen("command", "r")
```

45740 to start *command*, which is part of the same application. The parent writes a prompt to its  
45741 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child  
45742 reads the response from the user, does some transformation on the response (pathname  
45743 expansion, perhaps) and writes the result to its standard output. The parent process reads the  
45744 result from the pipe, does something with it, and prints another prompt. The cycle repeats.  
45745 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

45746 To conform to POSIX.1-2008, *pclose()* must use *waitpid()*, or some similar function, instead of  
45747 *wait()*.

45748 The code sample below illustrates how the *pclose()* function might be implemented on a system  
45749 conforming to POSIX.1-2008.

```
45750 int pclose(FILE *stream)
45751 {
45752     int stat;
45753     pid_t pid;
45754
45755     pid = <pid for process created for stream by popen()>
45756     (void) fclose(stream);
45757     while (waitpid(pid, &stat, 0) == -1) {
45758         if (errno != EINTR) {
45759             stat = -1;
45760             break;
45761         }
45762     }
45763     return(stat);
45764 }
```

#### 45764 FUTURE DIRECTIONS

45765 None.

#### 45766 SEE ALSO

45767 [fork\(\)](#), [popen\(\)](#), [wait\(\)](#)

45768 XBD [<stdio.h>](#)

#### 45769 CHANGE HISTORY

45770 First released in Issue 1. Derived from Issue 1 of the SVID.

**perror()**45771 **NAME**45772            **perror** — write error messages to standard error45773 **SYNOPSIS**

45774            #include &lt;stdio.h&gt;

45775            void perror(const char \*s);

45776 **DESCRIPTION**45777 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
45778 conflict between the requirements described here and the ISO C standard is unintentional. This  
45779 volume of POSIX.1-2008 defers to the ISO C standard.45780        The *perror()* function shall map the error number accessed through the symbol *errno* to a  
45781 language-dependent error message, which shall be written to the standard error stream as  
45782 follows:

- 45783
- First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the  
45784 string pointed to by *s* followed by a <colon> and a <space>.
  - Then an error message string followed by a <newline>.

45786        The contents of the error message strings shall be the same as those returned by *strerror()* with  
45787 argument *errno*.45788 **CX**        The *perror()* function shall mark for update the last data modification and last file status change  
45789 timestamps of the file associated with the standard error stream at some time between its  
45790 successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.45791        The *perror()* function shall not change the orientation of the standard error stream.45792 **RETURN VALUE**45793        The *perror()* function shall not return a value.45794 **ERRORS**

45795        No errors are defined.

45796 **EXAMPLES**45797            **Printing an Error Message for a Function**45798        The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,  
45799 the *perror()* function prints a message and the program exits.45800        #include <stdio.h>  
45801        #include <stdlib.h>  
45802        ...  
45803        char \*bufptr;  
45804        size\_t szbuf;  
45805        ...  
45806        if ((bufptr = malloc(szbuf)) == NULL) {  
45807            perror("malloc"); exit(2);  
45808        }  
45809        ...45810 **APPLICATION USAGE**

45811        None.

45812 **RATIONALE**

45813 None.

45814 **FUTURE DIRECTIONS**

45815 None.

45816 **SEE ALSO**45817 *psiginfo()*, *strerror()*

45818 XBD &lt;stdio.h&gt;

45819 **CHANGE HISTORY**

45820 First released in Issue 1. Derived from Issue 1 of the SVID.

45821 **Issue 5**45822 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the  
45823 orientation of the standard error stream.45824 **Issue 6**

45825 Extensions beyond the ISO C standard are marked.

45826 **Issue 7**

45827 SD5-XSH-ERN-95 is applied.

45828 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pipe()**45829 **NAME**

45830 pipe — create an interprocess channel

45831 **SYNOPSIS**

```
45832 #include <unistd.h>
45833 int pipe(int fildes[2]);
```

45834 **DESCRIPTION**

45835 The *pipe()* function shall create a pipe and place two file descriptors, one each into the  
 45836 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write  
 45837 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*  
 45838 call. The *O\_NONBLOCK* and *FD\_CLOEXEC* flags shall be clear on both file descriptors. (The  
 45839 *fcntl()* function can be used to set both these flags.)

45840 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A  
 45841 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a  
 45842 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether  
 45843 *fildes*[1] is also open for reading.

45844 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open  
 45845 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

45846 The pipe's user ID shall be set to the effective user ID of the calling process.

45847 The pipe's group ID shall be set to the effective group ID of the calling process.

45848 Upon successful completion, *pipe()* shall mark for update the last data access, last data  
 45849 modification, and last file status change timestamps of the pipe.

45850 **RETURN VALUE**

45851 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 45852 indicate the error.

45853 **ERRORS**

45854 The *pipe()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 45855 | [EMFILE] | All, or all but one, of the file descriptors available to the process are currently open.  |
| 45857 | [ENFILE] | The number of simultaneously open files in the system would exceed a system-imposed limit. |

45859 **EXAMPLES**45860 **Using a Pipe to Pass Data Between a Parent Process and a Child Process**

45861 The following example demonstrates the use of a pipe to transfer data between a parent process  
 45862 and a child process. Error handling is excluded, but otherwise this code demonstrates good  
 45863 practice when using pipes: after the *fork()* the two processes close the unused ends of the pipe  
 45864 before they commence transferring data.

```
45865 #include <stdlib.h>
45866 #include <unistd.h>
45867 ...
45868 int fildes[2];
45869 const int BSIZE = 100;
45870 char buf[BSIZE];
45871 ssize_t nbytes;
```

```

45872     int status;
45873     status = pipe(fildes);
45874     if (status == -1 ) {
45875         /* an error occurred */
45876         ...
45877     }
45878     switch (fork()) {
45879     case -1: /* Handle error */
45880         break;
45881     case 0: /* Child - reads from pipe */
45882         close(fildes[1]); /* Write end is unused */
45883         nbytes = read(fildes[0], buf, BSIZE); /* Get data from pipe */
45884         /* At this point, a further read would see end of file ... */
45885         close(fildes[0]); /* Finished with pipe */
45886         exit(EXIT_SUCCESS);
45887     default: /* Parent - writes to pipe */
45888         close(fildes[0]); /* Read end is unused */
45889         write(fildes[1], "Hello world\n", 12); /* Write data on pipe */
45890         close(fildes[1]); /* Child will see EOF */
45891         exit(EXIT_SUCCESS);
45892     }

```

**45893 APPLICATION USAGE**

45894 None.

**45895 RATIONALE**

45896 The wording carefully avoids using the verb "to open" in order to avoid any implication of use  
45897 of *open()*; see also *write()*.

**45898 FUTURE DIRECTIONS**

45899 None.

**45900 SEE ALSO**

45901 *fcntl()*, *read()*, *write()*  
45902 XBD <*fcntl.h*>, <*unistd.h*>

**45903 CHANGE HISTORY**

45904 First released in Issue 1. Derived from Issue 1 of the SVID.

**45905 Issue 6**

45906 The following new requirements on POSIX implementations derive from alignment with the  
45907 Single UNIX Specification:

- 45908 • The DESCRIPTION is updated to indicate that certain dispositions of *fildes[0]* and *fildes[1]*  
45909 are unspecified.

45910 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the  
45911 EXAMPLES section.

## pipe()

45912 **Issue 7**

45913 SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user  
45914 ID and group ID.

45915 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 45916 NAME

45917 poll — input/output multiplexing

## 45918 SYNOPSIS

45919 #include &lt;poll.h&gt;

45920 int poll(struct pollfd *fds*[], nfd\_t *nfds*, int *timeout*);

## 45921 DESCRIPTION

45922 The *poll()* function provides applications with a mechanism for multiplexing input/output over  
 45923 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the  
 45924 given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the  
 45925 *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an  
 45926 application can read or write data, or on which certain events have occurred.

45927 The *fds* argument specifies the file descriptors to be examined and the events of interest for each  
 45928 file descriptor. It is a pointer to an array with one member for each open file descriptor of  
 45929 interest. The array's members are **pollfd** structures within which *fd* specifies an open file  
 45930 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the  
 45931 following event flags:

45932	POLLIN	Data other than high-priority data may be read without blocking.
45933	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
45934		This flag shall be equivalent to <code>POLLRDNORM   POLLRDBAND</code> .
45935	POLLRDNORM	Normal data may be read without blocking.
45936	OB XSR	For STREAMS, data on priority band 0 may be read without blocking. This
45937		flag is set in <i>revents</i> even if the message is of zero length.
45938	POLLRDBAND	Priority data may be read without blocking.
45939	OB XSR	For STREAMS, data on priority bands greater than 0 may be read without
45940		blocking. This flag is set in <i>revents</i> even if the message is of zero length.
45941	POLLPRI	High-priority data may be read without blocking.
45942	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
45943	POLLOUT	Normal data may be written without blocking.
45944	OB XSR	For STREAMS, data on priority band 0 may be written without blocking.
45945	POLLWRNORM	Equivalent to POLLOUT.
45946	POLLWRBAND	Priority data may be written.
45947	OB XSR	For STREAMS, data on priority bands greater than 0 may be written without
45948		blocking. If any priority band has been written to on this STREAM, this event
45949		only examines bands that have been written to at least once.
45950	POLLERR	An error has occurred on the device or stream. This flag is only valid in the
45951		<i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.
45952	POLLHUP	A device has been disconnected, or a pipe or FIFO has been closed by the last
45953		process that had it open for writing. Once set, the hangup state of a FIFO shall
45954		persist until some process opens the FIFO for writing or until all read-only file
45955		descriptors for the FIFO are closed. This event and POLLOUT are mutually-
45956		exclusive; a stream can never be writable if a hangup has occurred. However,
45957		this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are
45958		not mutually-exclusive. This flag is only valid in the <i>revents</i> bitmask; it shall be

**poll()**

45959		ignored in the <i>events</i> member.
45960	POLLNVAL	The specified <i>fd</i> value is invalid. This flag is only valid in the <i>revents</i> member; it shall ignored in the <i>events</i> member.
45961		
45962		The significance and semantics of normal, priority, and high-priority data are file and device-specific.
45963		
45964		If the value of <i>fd</i> is less than 0, <i>events</i> shall be ignored, and <i>revents</i> shall be set to 0 in that entry on return from <i>poll()</i> .
45965		
45966		In each <b>pollfd</b> structure, <i>poll()</i> shall clear the <i>revents</i> member, except that where the application requested a report on a condition by setting one of the bits of <i>events</i> listed above, <i>poll()</i> shall set the corresponding bit in <i>revents</i> if the requested condition is true. In addition, <i>poll()</i> shall set the POLLHUP, POLLERR, and POLLNVAL flag in <i>revents</i> if the condition is true, even if the application did not set the corresponding bit in <i>events</i> .
45967		
45968		
45969		
45970		
45971		If none of the defined events have occurred on any selected file descriptor, <i>poll()</i> shall wait at least <i>timeout</i> milliseconds for an event to occur on any of the selected file descriptors. If the value of <i>timeout</i> is 0, <i>poll()</i> shall return immediately. If the value of <i>timeout</i> is -1, <i>poll()</i> shall block until a requested event occurs or until the call is interrupted.
45972		
45973		
45974		
45975		Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.
45976		
45977		
45978		The <i>poll()</i> function shall not be affected by the O_NONBLOCK flag.
45979		The <i>poll()</i> function shall support regular files, terminal and pseudo-terminal devices, FIFOs,
45980	OB XSR	pipes, sockets and <b>STREAMS-based files</b> . The behavior of <i>poll()</i> on elements of <i>fds</i> that refer to other types of file is unspecified.
45981		
45982		Regular files shall always poll TRUE for reading and writing.
45983		A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.
45984		
45985		
45986		<b>RETURN VALUE</b>
45987		Upon successful completion, <i>poll()</i> shall return a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the <i>revents</i> member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, <i>poll()</i> shall return -1 and set <i>errno</i> to indicate the error.
45988		
45989		
45990		
45991		<b>ERRORS</b>
45992		The <i>poll()</i> function shall fail if:
45993	[EAGAIN]	The allocation of internal data structures failed but a subsequent request may succeed.
45994		
45995	[EINTR]	A signal was caught during <i>poll()</i> .
45996	OB XSR [EINVAL]	The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members refers to a <b>STREAM</b> or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.
45997		
45998		

## 45999 EXAMPLES

46000 **Checking for Events on a Stream**

46001 The following example opens a pair of STREAMS devices and then waits for either one to  
 46002 become writable. This example proceeds as follows:

- 46003 1. Sets the *timeout* parameter to 500 milliseconds.
- 46004 2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying  
 46005 POLLOUT and POLLWRBAND as the events of interest.

46006 The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how  
 46007 STREAMS devices can be named; STREAMS naming conventions may vary among  
 46008 systems conforming to the POSIX.1-2008.

- 46009 3. Uses the *ret* variable to determine whether an event has occurred on either of the two  
 46010 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it  
 46011 has not occurred prior to the *poll()* call).
- 46012 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can  
 46013 be done:
  - 46014 a. Priority data can be written to the open STREAM on priority bands greater than 0,  
 46015 because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
  - 46016 b. Data can be written to the open STREAM on priority-band 0, because the  
 46017 POLLOUT event occurred on the open STREAM (*fds[0]* or *fds[1]*).
- 46018 5. If the returned value is not a positive value, permission to write data to the open  
 46019 STREAM (on any priority band) is denied.
- 46020 6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the  
 46021 open STREAM has disconnected.

```

46022 #include <stropts.h>
46023 #include <poll.h>
46024 ...
46025 struct pollfd fds[2];
46026 int timeout_msecs = 500;
46027 int ret;
46028     int i;

46029 /* Open STREAMS device. */
46030 fds[0].fd = open("/dev/dev0", ...);
46031 fds[1].fd = open("/dev/dev1", ...);
46032 fds[0].events = POLLOUT | POLLWRBAND;
46033 fds[1].events = POLLOUT | POLLWRBAND;

46034 ret = poll(fds, 2, timeout_msecs);

46035 if (ret > 0) {
46036     /* An event on one of the fds has occurred. */
46037     for (i=0; i<2; i++) {
46038         if (fds[i].revents & POLLWRBAND) {
46039             /* Priority data may be written on device number i. */
46040             ...
46041         }
46042         if (fds[i].revents & POLLOUT) {

```

**poll()**

```

46043             /* Data may be written on device number i. */
46044     ...
46045     }
46046     if (fds[i].revents & POLLHUP) {
46047         /* A hangup has occurred on device number i. */
46048     ...
46049     }
46050     }
46051 }

```

**46052 APPLICATION USAGE**

46053 None.

**46054 RATIONALE**

46055 The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It  
46056 only occurs when the FIFO is closed by the last writer and persists until some process opens the  
46057 FIFO for writing or until all read-only file descriptors for the FIFO are closed.

**46058 FUTURE DIRECTIONS**

46059 None.

**46060 SEE ALSO**

46061 [Section 2.6](#) (on page 494), [getmsg\(\)](#), [pselect\(\)](#), [putmsg\(\)](#), [read\(\)](#), [write\(\)](#)

46062 XBD [<poll.h>](#), [<stropts.h>](#)

**46063 CHANGE HISTORY**

46064 First released in Issue 4, Version 2.

**46065 Issue 5**

46066 Moved from X/OPEN UNIX extension to BASE.

46067 The description of POLLWRBAND is updated.

**46068 Issue 6**

46069 Text referring to sockets is added to the DESCRIPTION.

46070 Functionality relating to the XSI STREAMS Option Group is marked.

46071 The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of  
46072 POLLWRBAND.

46073 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in  
46074 the EXAMPLES section.

**46075 Issue 7**

46076 Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

46077 The *poll()* function is moved from the XSI option to the Base.

46078 Functionality relating to the XSI STREAMS option is marked obsolescent.

46079 **NAME**

46080 popen — initiate pipe streams to or from a process

46081 **SYNOPSIS**

```
46082 CX #include <stdio.h>
46083 FILE *popen(const char *command, const char *mode);
```

46084 **DESCRIPTION**

46085 The *popen()* function shall execute the command specified by the string *command*. It shall create  
 46086 a pipe between the calling program and the executed command, and shall return a pointer to a  
 46087 stream that can be used to either read from or write to the pipe.

46088 The environment of the executed command shall be as if a child process were created within the  
 46089 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

```
46090 execl(shell_path, "sh", "-c", command, (char *)0);
```

46091 where *shell\_path* is an unspecified pathname for the *sh* utility.

46092 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open  
 46093 in the parent process are closed in the new child process.

46094 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 46095 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be  
 46096 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 46097 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the  
 46098 pipe.
- 46099 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be  
 46100 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 46101 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the  
 46102 pipe.
- 46103 3. If *mode* is any other value, the result is unspecified.

46104 After *popen()*, both the parent and the child process shall be capable of executing independently  
 46105 before either terminates.

46106 Pipe streams are byte-oriented.

46107 **RETURN VALUE**

46108 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to  
 46109 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate  
 46110 the error.

46111 **ERRORS**

46112 The *popen()* function shall fail if:

46113 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

46114 The *popen()* function may fail if:

46115 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

46116 [EINVAL] The *mode* argument is invalid.

46117 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

**popen()**46118 **EXAMPLES**46119 **Using popen() to Obtain a List of Files from the ls Utility**

46120 The following example demonstrates the use of *popen()* and *pclose()* to execute the command *ls\**  
 46121 in order to obtain a list of files in the current directory:

```

46122 #include <stdio.h>
46123 ...
46124 FILE *fp;
46125 int status;
46126 char path[PATH_MAX];
46127 fp = popen("ls *", "r");
46128 if (fp == NULL)
46129     /* Handle error */;
46130 while (fgets(path, PATH_MAX, fp) != NULL)
46131     printf("%s", path);
46132 status = pclose(fp);
46133 if (status == -1) {
46134     /* Error reported by pclose() */
46135     ...
46136 } else {
46137     /* Use macros described under wait() to inspect 'status' in order
46138     to determine success/failure of command executed by popen() */
46139     ...
46140 }

```

46141 **APPLICATION USAGE**

46142 Since open files are shared, a mode *r* command can be used as an input filter and a mode *w*  
 46143 command as an output filter.

46144 Buffered reading before opening an input filter may leave the standard input of that filter  
 46145 mispositioned. Similar problems with an output filter may be prevented by careful buffer  
 46146 flushing; for example, with *fflush()*.

46147 A stream opened by *popen()* should be closed by *pclose()*.

46148 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and  
 46149 *wb* might be supported by specific implementations, but these would not be portable features.  
 46150 Note that historical implementations of *popen()* only check to see if the first character of *mode* is  
 46151 *r*. Thus, a mode of *robert the robot* would be treated as mode *r*, and a mode of *anything else* would be  
 46152 treated as mode *w*.

46153 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a  
 46154 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process  
 46155 started by *popen()*.

46156 To determine whether or not the environment specified in the Shell and Utilities volume of  
 46157 POSIX.1-2008 is present, use the function call:

```
46158 sysconf (_SC_2_VERSION)
```

46159 (See *sysconf()*).

46160 **RATIONALE**

46161 The *popen()* function should not be used by programs that have set user (or group) ID privileges.  
 46162 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.  
 46163 This prevents any unforeseen manipulation of the environment of the user that could cause  
 46164 execution of commands not anticipated by the calling program.

46165 If the original and *popen()*ed processes both intend to read or write or read and write a common  
 46166 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for  
 46167 sharing file handles must be observed (see Section 2.5.1, on page 491).

46168 **FUTURE DIRECTIONS**

46169 None.

46170 **SEE ALSO**

46171 *fork()*, *pclose()*, *pipe()*, *sysconf()*, *system()*, *wait()*, *waitid()*

46172 XBD <stdio.h>

46173 XCU *sh*

46174 **CHANGE HISTORY**

46175 First released in Issue 1. Derived from Issue 1 of the SVID.

46176 **Issue 5**

46177 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

46178 **Issue 6**

46179 The following new requirements on POSIX implementations derive from alignment with the  
 46180 Single UNIX Specification:

- 46181 • The optional [EMFILE] error condition is added.

46182 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the  
 46183 EXAMPLES section.

46184 **Issue 7**

46185 Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the  
 46186 DESCRIPTION.

46187 SD5-XSH-ERN-149 is applied, changing the {STREAM\_MAX} [EMFILE] error condition from a  
 46188 "may fail" to a "shall fail".

**posix\_fadvise()**46189 **NAME**46190 `posix_fadvise` — file advisory information (**ADVANCED REALTIME**)46191 **SYNOPSIS**

```
46192 ADV #include <fcntl.h>
46193 int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

46194 **DESCRIPTION**

46195 The `posix_fadvise()` function shall advise the implementation on the expected behavior of the  
 46196 application with respect to the data in the file associated with the open file descriptor `fd`, starting  
 46197 at `offset` and continuing for `len` bytes. The specified range need not currently exist in the file. If `len`  
 46198 is zero, all data following `offset` is specified. The implementation may use this information to  
 46199 optimize handling of the specified data. The `posix_fadvise()` function shall have no effect on the  
 46200 semantics of other operations on the specified data, although it may affect the performance of  
 46201 other operations.

46202 The advice to be applied to the data is specified by the `advice` parameter and may be one of the  
 46203 following values:

46204 **POSIX\_FADV\_NORMAL**

46205 Specifies that the application has no advice to give on its behavior with respect to the  
 46206 specified data. It is the default characteristic if no advice is given for an open file.

46207 **POSIX\_FADV\_SEQUENTIAL**

46208 Specifies that the application expects to access the specified data sequentially from lower  
 46209 offsets to higher offsets.

46210 **POSIX\_FADV\_RANDOM**

46211 Specifies that the application expects to access the specified data in a random order.

46212 **POSIX\_FADV\_WILLNEED**

46213 Specifies that the application expects to access the specified data in the near future.

46214 **POSIX\_FADV\_DONTNEED**

46215 Specifies that the application expects that it will not access the specified data in the near  
 46216 future.

46217 **POSIX\_FADV\_NOREUSE**

46218 Specifies that the application expects to access the specified data once and then not reuse it  
 46219 thereafter.

46220 These values are defined in `<fcntl.h>`.

46221 **RETURN VALUE**

46222 Upon successful completion, `posix_fadvise()` shall return zero; otherwise, an error number shall  
 46223 be returned to indicate the error.

46224 **ERRORS**

46225 The `posix_fadvise()` function shall fail if:

- |       |          |  |
|-------|----------|--|
| 46226 | [EBADF]  | The <code>fd</code> argument is not a valid file descriptor.                                     |
| 46227 | [EINVAL] | The value of <code>advice</code> is invalid, or the value of <code>len</code> is less than zero. |
| 46228 | [ESPIPE] | The <code>fd</code> argument is associated with a pipe or FIFO.                                  |

**46229 EXAMPLES**

46230 None.

**46231 APPLICATION USAGE**

46232 The *posix\_fadvise()* function is part of the Advisory Information option and need not be  
46233 provided on all implementations.

**46234 RATIONALE**

46235 None.

**46236 FUTURE DIRECTIONS**

46237 None.

**46238 SEE ALSO**

46239 *posix\_madvise()*

46240 XBD `<fcntl.h>`

**46241 CHANGE HISTORY**

46242 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46243 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46244 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function  
46245 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the  
46246 standard developers felt it acceptable to make this change before implementations of this  
46247 function become widespread.

**46248 Issue 7**

46249 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the  
46250 [EINVAL] error.

**posix\_fallocate()**46251 **NAME**46252 `posix_fallocate` — file space control (**ADVANCED REALTIME**)46253 **SYNOPSIS**

```
46254 ADV #include <fcntl.h>
46255 int posix_fallocate(int fd, off_t offset, off_t len);
```

46256 **DESCRIPTION**

46257 The `posix_fallocate()` function shall ensure that any required storage for regular file data starting  
 46258 at `offset` and continuing for `len` bytes is allocated on the file system storage media. If  
 46259 `posix_fallocate()` returns successfully, subsequent writes to the specified file data shall not fail due  
 46260 to the lack of free space on the file system storage media.

46261 If the `offset+len` is beyond the current file size, then `posix_fallocate()` shall adjust the file size to  
 46262 `offset+len`. Otherwise, the file size shall not be changed.

46263 It is implementation-defined whether a previous `posix_fadvise()` call influences allocation  
 46264 strategy.

46265 Space allocated via `posix_fallocate()` shall be freed by a successful call to `creat()` or `open()` that  
 46266 truncates the size of the file. Space allocated via `posix_fallocate()` may be freed by a successful call  
 46267 to `ftruncate()` that reduces the file size to a size smaller than `offset+len`.

46268 **RETURN VALUE**

46269 Upon successful completion, `posix_fallocate()` shall return zero; otherwise, an error number shall  
 46270 be returned to indicate the error.

46271 **ERRORS**46272 The `posix_fallocate()` function shall fail if:46273 [EBADF] The `fd` argument is not a valid file descriptor.46274 [EBADF] The `fd` argument references a file that was opened without write permission.46275 [EFBIG] The value of `offset+len` is greater than the maximum file size.

46276 [EINTR] A signal was caught during execution.

46277 [EINVAL] The `len` argument is less than zero, or the `offset` argument is less than zero, or  
 46278 the underlying file system does not support this operation.

46279 [EIO] An I/O error occurred while reading from or writing to a file system.

46280 [ENODEV] The `fd` argument does not refer to a regular file.

46281 [ENOSPC] There is insufficient free space remaining on the file system storage media.

46282 [ESPIPE] The `fd` argument is associated with a pipe or FIFO.46283 The `posix_fallocate()` function may fail if:46284 [EINVAL] The `len` argument is zero.

46285 **EXAMPLES**

46286 None.

46287 **APPLICATION USAGE**46288 The *posix\_fallocate()* function is part of the Advisory Information option and need not be  
46289 provided on all implementations.46290 **RATIONALE**

46291 None.

46292 **FUTURE DIRECTIONS**

46293 None.

46294 **SEE ALSO**46295 *creat()*, *ftruncate()*, *open()*, *unlink()*46296 XBD `<fcntl.h>`46297 **CHANGE HISTORY**

46298 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46299 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.46300 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function  
46301 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the  
46302 standard developers felt it acceptable to make this change before implementations of this  
46303 function become widespread.46304 **Issue 7**46305 Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the  
46306 definition of the [EINVAL] error.

**posix\_madvise()**

System Interfaces

46307 **NAME**

46308 **posix\_madvise** — memory advisory information and alignment control (**ADVANCED**  
 46309 **REALTIME**)

46310 **SYNOPSIS**

```
46311 ADV #include <sys/mman.h>
46312 int posix_madvise(void *addr, size_t len, int advice);
```

46313 **DESCRIPTION**

46314 The *posix\_madvise()* function shall advise the implementation on the expected behavior of the  
 46315 application with respect to the data in the memory starting at address *addr*, and continuing for  
 46316 *len* bytes. The implementation may use this information to optimize handling of the specified  
 46317 data. The *posix\_madvise()* function shall have no effect on the semantics of access to memory in  
 46318 the specified range, although it may affect the performance of access.

46319 The implementation may require that *addr* be a multiple of the page size, which is the value  
 46320 returned by *sysconf()* when the name value *\_SC\_PAGESIZE* is used.

46321 The advice to be applied to the memory range is specified by the *advice* parameter and may be  
 46322 one of the following values:

46323 **POSIX\_MADV\_NORMAL**

46324 Specifies that the application has no advice to give on its behavior with respect to the  
 46325 specified range. It is the default characteristic if no advice is given for a range of memory.

46326 **POSIX\_MADV\_SEQUENTIAL**

46327 Specifies that the application expects to access the specified range sequentially from lower  
 46328 addresses to higher addresses.

46329 **POSIX\_MADV\_RANDOM**

46330 Specifies that the application expects to access the specified range in a random order.

46331 **POSIX\_MADV\_WILLNEED**

46332 Specifies that the application expects to access the specified range in the near future.

46333 **POSIX\_MADV\_DONTNEED**

46334 Specifies that the application expects that it will not access the specified range in the near  
 46335 future.

46336 These values are defined in the `<sys/mman.h>` header.

46337 **RETURN VALUE**

46338 Upon successful completion, *posix\_madvise()* shall return zero; otherwise, an error number shall  
 46339 be returned to indicate the error.

46340 **ERRORS**

46341 The *posix\_madvise()* function shall fail if:

46342 [EINVAL] The value of *advice* is invalid.

46343 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly  
 46344 or completely outside the range allowed for the address space of the calling  
 46345 process.

- 46346 The *posix\_madvise()* function may fail if:
- 46347 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the  
46348 name value *\_SC\_PAGESIZE* is used.
- 46349 [EINVAL] The value of *len* is zero.
- 46350 **EXAMPLES**
- 46351 None.
- 46352 **APPLICATION USAGE**
- 46353 The *posix\_madvise()* function is part of the Advisory Information option and need not be  
46354 provided on all implementations.
- 46355 **RATIONALE**
- 46356 None.
- 46357 **FUTURE DIRECTIONS**
- 46358 None.
- 46359 **SEE ALSO**
- 46360 *mmap()*, *posix\_fadvise()*, *sysconf()*
- 46361 XBD <[sys/mman.h](#)>
- 46362 **CHANGE HISTORY**
- 46363 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 46364 IEEE PASC Interpretation 1003.1 #102 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_mem\_offset()**

System Interfaces

46365 **NAME**

46366 **posix\_mem\_offset** — find offset and length of a mapped typed memory block (**ADVANCED**  
 46367 **REALTIME**)

46368 **SYNOPSIS**

```
46369 TYM #include <sys/mman.h>
46370 int posix_mem_offset(const void *restrict addr, size_t len,
46371 off_t *restrict off, size_t *restrict contig_len,
46372 int *restrict fildes);
```

46373 **DESCRIPTION**

46374 The *posix\_mem\_offset()* function shall return in the variable pointed to by *off* a value that  
 46375 identifies the offset (or location), within a memory object, of the memory block currently  
 46376 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used  
 46377 (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since  
 46378 the mapping was established, the returned value of *fildes* shall be  $-1$ . The *len* argument specifies  
 46379 the length of the block of the memory object the user wishes the offset for; upon return, the  
 46380 value pointed to by *contig\_len* shall equal either *len*, or the length of the largest contiguous block  
 46381 of the memory object that is currently mapped to the calling process starting at *addr*, whichever  
 46382 is smaller.

46383 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig\_len*  
 46384 values obtained by calling *posix\_mem\_offset()* are used in a call to *mmap()* with a file descriptor  
 46385 that refers to the same memory pool as *fildes* (either through the same port or through a different  
 46386 port), and that was opened with neither the **POSIX\_TYPED\_MEM\_ALLOCATE** nor the  
 46387 **POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG** flag, the typed memory area that is mapped shall  
 46388 be exactly the same area that was mapped at *addr* in the address space of the process that called  
 46389 *posix\_mem\_offset()*.

46390 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
 46391 function is implementation-defined.

46392 **RETURN VALUE**

46393 Upon successful completion, the *posix\_mem\_offset()* function shall return zero; otherwise, the  
 46394 corresponding error status value shall be returned.

46395 **ERRORS**

46396 The *posix\_mem\_offset()* function shall fail if:

46397 [EACCES] The process has not mapped a memory object supported by this function at  
 46398 the given address *addr*.

46399 This function shall not return an error code of [EINTR].

46400 **EXAMPLES**

46401 None.

46402 **APPLICATION USAGE**

46403 None.

46404 **RATIONALE**

46405 None.

46406 **FUTURE DIRECTIONS**

46407 None.

46408 **SEE ALSO**46409 [mmap\(\)](#), [posix\\_typed\\_mem\\_open\(\)](#)46410 XBD [<sys/mman.h>](#)46411 **CHANGE HISTORY**

46412 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_memalign()**46413 **NAME**46414 posix\_memalign — aligned memory allocation (**ADVANCED REALTIME**)46415 **SYNOPSIS**

```
46416 ADV #include <stdlib.h>
46417 int posix_memalign(void **memptr, size_t alignment, size_t size);
```

46418 **DESCRIPTION**

46419 The *posix\_memalign()* function shall allocate *size* bytes aligned on a boundary specified by  
 46420 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*  
 46421 shall be a power of two multiple of *sizeof(void \*)*.

46422 Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

46423 If the size of the space requested is 0, the behavior is implementation-defined; the value returned  
 46424 in *memptr* shall be either a null pointer or a unique pointer.

46425 CX The *free()* function shall deallocate memory that has previously been allocated by  
 46426 *posix\_memalign()*.

46427 **RETURN VALUE**

46428 Upon successful completion, *posix\_memalign()* shall return zero; otherwise, an error number  
 46429 shall be returned to indicate the error.

46430 **ERRORS**

46431 The *posix\_memalign()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 46432 | [EINVAL] | The value of the alignment parameter is not a power of two multiple of <i>sizeof(void *)</i> . |
| 46433 |          |  |
| 46434 | [ENOMEM] | There is insufficient memory available with the requested alignment.                           |

46435 **EXAMPLES**

46436 None.

46437 **APPLICATION USAGE**

46438 The *posix\_memalign()* function is part of the Advisory Information option and need not be  
 46439 provided on all implementations.

46440 **RATIONALE**

46441 None.

46442 **FUTURE DIRECTIONS**

46443 None.

46444 **SEE ALSO**

46445 *free()*, *malloc()*

46446 XBD <stdlib.h>

46447 **CHANGE HISTORY**

46448 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46449 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

46450 **Issue 7**

46451 Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment*  
 46452 argument in the DESCRIPTION.

46453  
46454

Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of the space requested is 0.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_openpt()**46455 **NAME**46456 `posix_openpt` — open a pseudo-terminal device46457 **SYNOPSIS**

```
46458 XSI #include <stdlib.h>
46459 #include <fcntl.h>
46460 int posix_openpt(int oflag);
```

46461 **DESCRIPTION**

46462 The `posix_openpt()` function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

46465 The file status flags and file access modes of the open file description shall be set according to the value of `oflag`.

46467 Values for `oflag` are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

46469 `O_RDWR` Open for reading and writing.

46470 `O_NOCTTY` If set `posix_openpt()` shall not cause the terminal device to become the controlling terminal for the process.

46472 The behavior of other values for the `oflag` argument is unspecified.

46473 **RETURN VALUE**

46474 Upon successful completion, the `posix_openpt()` function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, `-1` shall be returned and `errno` set to indicate the error.

46477 **ERRORS**

46478 The `posix_openpt()` function shall fail if:

46479 [EMFILE] All file descriptors available to the process are currently open.

46480 [ENFILE] The maximum allowable number of files is currently open in the system.

46481 The `posix_openpt()` function may fail if:

46482 [EINVAL] The value of `oflag` is not valid.

46483 [EAGAIN] Out of pseudo-terminal resources.

46484 OB XSR [ENOSR] Out of STREAMS resources.

46485 **EXAMPLES**

46486 **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor**

```
46488 #include <fcntl.h>
46489 #include <stdio.h>
46490 int masterfd, slavefd;
46491 char *slavedevice;
46492 masterfd = posix_openpt(O_RDWR|O_NOCTTY);
46493 if (masterfd == -1
46494     || grantpt(masterfd) == -1
```

```

46495         || unlockpt (masterfd) == -1
46496         || (slavedevice = ptsname (masterfd)) == NULL)
46497         return -1;

46498     printf("slave device is: %s\n", slavedevice);

46499     slavefd = open(slavedevice, O_RDWR|O_NOCTTY);
46500     if (slavefd < 0)
46501         return -1;

```

#### 46502 APPLICATION USAGE

46503 This function is a method for portably obtaining a file descriptor of a master terminal device for  
 46504 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and  
 46505 ownership permissions, and to obtain the name of the slave device, respectively.

#### 46506 RATIONALE

46507 The standard developers considered the matter of adding a special device for cloning master  
 46508 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was  
 46509 felt that adding a new function would permit other implementations. The *posix\_openpt()*  
 46510 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

46511 On implementations supporting the */dev/ptmx* clone device opening the master device of a  
 46512 pseudo-terminal is simply:

```

46513 mfdp = open("/dev/ptmx", oflag );
46514 if (mfdp < 0)
46515     return -1;

```

#### 46516 FUTURE DIRECTIONS

46517 None.

#### 46518 SEE ALSO

46519 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*

46520 XBD <*fcntl.h*>, <*stdlib.h*>

#### 46521 CHANGE HISTORY

46522 First released in Issue 6

#### 46523 Issue 7

46524 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

46525 SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

**posix\_spawn()**46526 **NAME**46527 `posix_spawn`, `posix_spawnp` — spawn a process (**ADVANCED REALTIME**)46528 **SYNOPSIS**

```

46529 SPN #include <spawn.h>
46530 int posix_spawn(pid_t *restrict pid, const char *restrict path,
46531               const posix_spawn_file_actions_t *file_actions,
46532               const posix_spawnattr_t *restrict attrp,
46533               char *const argv[restrict], char *const envp[restrict]);
46534 int posix_spawnp(pid_t *restrict pid, const char *restrict file,
46535                const posix_spawn_file_actions_t *file_actions,
46536                const posix_spawnattr_t *restrict attrp,
46537                char *const argv[restrict], char *const envp[restrict]);

```

46538 **DESCRIPTION**

46539 The `posix_spawn()` and `posix_spawnp()` functions shall create a new process (child process) from  
 46540 the specified process image. The new process image shall be constructed from a regular  
 46541 executable file called the new process image file.

46542 When a C program is executed as the result of this call, it shall be entered as a C-language  
 46543 function call as follows:

```
46544 int main(int argc, char *argv[]);
```

46545 where `argc` is the argument count and `argv` is an array of character pointers to the arguments  
 46546 themselves. In addition, the following variable:

```
46547 extern char **environ;
```

46548 shall be initialized as a pointer to an array of character pointers to the environment strings.

46549 The argument `argv` is an array of character pointers to null-terminated strings. The last member  
 46550 of this array shall be a null pointer and is not counted in `argc`. These strings constitute the  
 46551 argument list available to the new process image. The value in `argv[0]` should point to a filename  
 46552 that is associated with the process image being started by the `posix_spawn()` or `posix_spawnp()`  
 46553 function.

46554 The argument `envp` is an array of character pointers to null-terminated strings. These strings  
 46555 constitute the environment for the new process image. The environment array is terminated by a  
 46556 null pointer.

46557 The number of bytes available for the combined argument and environment lists of the child  
 46558 process is {ARG\_MAX}. The implementation shall specify in the system documentation (see  
 46559 XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators,  
 46560 pointers, or alignment bytes, is included in this total.

46561 The `path` argument to `posix_spawn()` is a pathname that identifies the new process image file to  
 46562 execute.

46563 The `file` parameter to `posix_spawnp()` shall be used to construct a pathname that identifies the  
 46564 new process image file. If the `file` parameter contains a <slash> character, the `file` parameter shall  
 46565 be used as the pathname for the new process image file. Otherwise, the path prefix for this file  
 46566 shall be obtained by a search of the directories passed as the environment variable `PATH` (see  
 46567 XBD Chapter 8, on page 173). If this environment variable is not defined, the results of the  
 46568 search are implementation-defined.

46569 If `file_actions` is a null pointer, then file descriptors open in the calling process shall remain open

46570 in the child process, except for those whose close-on-exec flag FD\_CLOEXEC is set (see *fcntl()*).  
 46571 For those file descriptors that remain open, all attributes of the corresponding open file  
 46572 descriptions, including file locks (see *fcntl()*), shall remain unchanged.

46573 If *file\_actions* is not NULL, then the file descriptors open in the child process shall be those open  
 46574 in the calling process as modified by the spawn file actions object pointed to by *file\_actions* and  
 46575 the FD\_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have  
 46576 been processed. The effective order of processing the spawn file actions shall be:

- 46577 1. The set of open file descriptors for the child process shall initially be the same set as is  
 46578 open for the calling process. All attributes of the corresponding open file descriptions,  
 46579 including file locks (see *fcntl()*), shall remain unchanged.
- 46580 2. The signal mask, signal default actions, and the effective user and group IDs for the child  
 46581 process shall be changed as specified in the attributes object referenced by *attrp*.
- 46582 3. The file actions specified by the spawn file actions object shall be performed in the order  
 46583 in which they were added to the spawn file actions object.
- 46584 4. Any file descriptor that has its FD\_CLOEXEC flag set (see *fcntl()*) shall be closed.

46585 The **posix\_spawnattr\_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at  
 46586 least the attributes defined below.

46587 If the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of the object referenced  
 46588 by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process  
 46589 group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

46590 As a special case, if the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of  
 46591 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,  
 46592 then the child shall be in a new process group with a process group ID equal to its process ID.

46593 If the POSIX\_SPAWN\_SETPGROUP flag is not set in the *spawn\_flags* attribute of the object  
 46594 referenced by *attrp*, the new child process shall inherit the parent's process group.

46595 PS If the POSIX\_SPAWN\_SETSCHEDPARAM flag is set in the *spawn\_flags* attribute of the object  
 46596 referenced by *attrp*, but POSIX\_SPAWN\_SETSCHEDULER is not set, the new process image  
 46597 shall initially have the scheduling policy of the calling process with the scheduling parameters  
 46598 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

46599 If the POSIX\_SPAWN\_SETSCHEDULER flag is set in the *spawn\_flags* attribute of the object  
 46600 referenced by *attrp* (regardless of the setting of the POSIX\_SPAWN\_SETSCHEDPARAM flag),  
 46601 the new process image shall initially have the scheduling policy specified in the *spawn-*  
 46602 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in  
 46603 the *spawn-schedparam* attribute of the same object.

46604 The POSIX\_SPAWN\_RESETPIDS flag in the *spawn\_flags* attribute of the object referenced by *attrp*  
 46605 governs the effective user ID of the child process. If this flag is not set, the child process shall  
 46606 inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the  
 46607 child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit  
 46608 of the new process image file is set, the effective user ID of the child process shall become that  
 46609 file's owner ID before the new process image begins execution.

46610 The POSIX\_SPAWN\_RESETPIDS flag in the *spawn\_flags* attribute of the object referenced by *attrp*  
 46611 also governs the effective group ID of the child process. If this flag is not set, the child process  
 46612 shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID  
 46613 of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID  
 46614 mode bit of the new process image file is set, the effective group ID of the child process shall

**posix\_spawn()**

- 46615 become that file's group ID before the new process image begins execution.
- 46616 If the POSIX\_SPAWN\_SETSIGMASK flag is set in the *spawn\_flags* attribute of the object  
46617 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*  
46618 *sigmask* attribute of the object referenced by *attrp*.
- 46619 If the POSIX\_SPAWN\_SETSIGDEF flag is set in the *spawn\_flags* attribute of the object referenced  
46620 by *attrp*, the signals specified in the *spawn\_sigdefault* attribute of the same object shall be set to  
46621 their default actions in the child process. Signals set to the default action in the parent process  
46622 shall be set to the default action in the child process.
- 46623 Signals set to be caught by the calling process shall be set to the default action in the child  
46624 process.
- 46625 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be  
46626 ignored by the child process, unless otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag  
46627 being set in the *spawn\_flags* attribute of the object referenced by *attrp* and the signals being  
46628 indicated in the *spawn\_sigdefault* attribute of the object referenced by *attrp*.
- 46629 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the  
46630 SIGCHLD signal is set to be ignored or to the default action in the child process, unless  
46631 otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag being set in the *spawn\_flags*  
46632 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the  
46633 *spawn\_sigdefault* attribute of the object referenced by *attrp*.
- 46634 If the value of the *attrp* pointer is NULL, then the default values are used.
- 46635 All process attributes, other than those influenced by the attributes set in the object referenced  
46636 by *attrp* as specified above or by the file descriptor manipulations specified in *file\_actions*, shall  
46637 appear in the new process image as though *fork()* had been called to create a child process and  
46638 then a member of the *exec* family of functions had been called by the child process to execute the  
46639 new process image.
- 46640 It is implementation-defined whether the fork handlers are run when *posix\_spawn()* or  
46641 *posix\_spawnp()* is called.
- 46642 **RETURN VALUE**
- 46643 Upon successful completion, *posix\_spawn()* and *posix\_spawnp()* shall return the process ID of the  
46644 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and  
46645 shall return zero as the function return value. Otherwise, no child process shall be created, the  
46646 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number  
46647 shall be returned as the function return value to indicate the error. If the *pid* argument is a null  
46648 pointer, the process ID of the child is not returned to the caller.
- 46649 **ERRORS**
- 46650 These functions may fail if:
- 46651 [EINVAL] The value specified by *file\_actions* or *attrp* is invalid.
- 46652 If this error occurs after the calling process successfully returns from the *posix\_spawn()* or  
46653 *posix\_spawnp()* function, the child process may exit with exit status 127.
- 46654 If *posix\_spawn()* or *posix\_spawnp()* fail for any of the reasons that would cause *fork()* or one of  
46655 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and  
46656 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child  
46657 process shall exit with exit status 127).
- 46658 If POSIX\_SPAWN\_SETPGROUP is set in the *spawn\_flags* attribute of the object referenced by  
46659 *attrp*, and *posix\_spawn()* or *posix\_spawnp()* fails while changing the child's process group, an

46660 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling  
46661 process successfully returns, the child process shall exit with exit status 127).

46662 PS If POSIX\_SPAWN\_SETSCHEDPARAM is set and POSIX\_SPAWN\_SETSCHEDULER is not set in  
46663 the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix\_spawn()* or *posix\_spawnp()*  
46664 fails for any of the reasons that would cause *sched\_setparam()* to fail, an error value shall be  
46665 returned as described by *sched\_setparam()* (or, if the error occurs after the calling process  
46666 successfully returns, the child process shall exit with exit status 127).

46667 If POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by  
46668 *attrp*, and if *posix\_spawn()* or *posix\_spawnp()* fails for any of the reasons that would cause  
46669 *sched\_setscheduler()* to fail, an error value shall be returned as described by *sched\_setscheduler()*  
46670 (or, if the error occurs after the calling process successfully returns, the child process shall exit  
46671 with exit status 127).

46672 If the *file\_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be  
46673 performed, and if *posix\_spawn()* or *posix\_spawnp()* fails for any of the reasons that would cause  
46674 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*,  
46675 and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the  
46676 child process shall exit with exit status 127). An open file action may, by itself, result in any of  
46677 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

#### 46678 EXAMPLES

46679 None.

#### 46680 APPLICATION USAGE

46681 These functions are part of the Spawn option and need not be provided on all implementations.

#### 46682 RATIONALE

46683 The *posix\_spawn()* function and its close relation *posix\_spawnp()* have been introduced to  
46684 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or  
46685 impossible to implement without swapping or dynamic address translation.

- 46686 • Swapping is generally too slow for a realtime environment.
- 46687 • Dynamic address translation is not available everywhere that POSIX might be useful.
- 46688 • Processes are too useful to simply option out of POSIX whenever it must run without  
46689 address translation or other MMU services.

46690 Thus, POSIX needs process creation and file execution primitives that can be efficiently  
46691 implemented without address translation or other MMU services.

46692 The *posix\_spawn()* function is implementable as a library routine, but both *posix\_spawn()* and  
46693 *posix\_spawnp()* are designed as kernel operations. Also, although they may be an efficient  
46694 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation  
46695 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for  
46696 *fork()/exec*.

46697 This view of the role of *posix\_spawn()* and *posix\_spawnp()* influenced the design of their API. It  
46698 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified  
46699 operations of any sort are permitted between the creation of the child process and the execution  
46700 of the new process image; any attempt to reach that level would need to provide a programming  
46701 language as parameters. Instead, *posix\_spawn()* and *posix\_spawnp()* are process creation  
46702 primitives like the *Start\_Process* and *Start\_Process\_Search* Ada language bindings package  
46703 *POSIX\_Process\_Primitives* and also like those in many operating systems that are not UNIX  
46704 systems, but with some POSIX-specific additions.

**posix\_spawn()**

46705 To achieve its coverage goals, *posix\_spawn()* and *posix\_spawnp()* have control of six types of  
 46706 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and  
 46707 whether each signal ignored in the parent will remain ignored in the child, or be reset to its  
 46708 default action in the child.

46709 Control of file descriptors is required to allow an independently written child process image to  
 46710 access data streams opened by and even generated or read by the parent process without being  
 46711 specifically coded to know which parent files and file descriptors are to be used. Control of the  
 46712 process group ID is required to control how the job control of the child process relates to that of  
 46713 the parent.

46714 Control of the signal mask and signal defaulting is sufficient to support the implementation of  
 46715 *system()*. Although support for *system()* is not explicitly one of the goals for *posix\_spawn()* and  
 46716 *posix\_spawnp()*, it is covered under the “at least 50%” coverage goal.

46717 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of  
 46718 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*  
 46719 family of functions should fully specify open file inheritance. The implementation need make no  
 46720 decisions regarding the set of open file descriptors when the child process image begins  
 46721 execution, those decisions having already been made by the caller and expressed as the set of  
 46722 open file descriptors and their FD\_CLOEXEC flags at the time of the call and the spawn file  
 46723 actions object specified in the call. We have been assured that in cases where the POSIX  
 46724 *Start\_Process* Ada primitives have been implemented in a library, this method of controlling file  
 46725 descriptor inheritance may be implemented very easily.

46726 We can identify several problems with *posix\_spawn()* and *posix\_spawnp()*, but there does not  
 46727 appear to be a solution that introduces fewer problems. Environment modification for child  
 46728 process attributes not specifiable via the *attrp* or *file\_actions* arguments must be done in the  
 46729 parent process, and since the parent generally wants to save its context, it is more costly than  
 46730 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a  
 46731 multi-threaded process temporarily, since all threads must agree when it is safe for the  
 46732 environment to be changed. However, this cost is only borne by those invocations of  
 46733 *posix\_spawn()* and *posix\_spawnp()* that use the additional functionality. Since extensive  
 46734 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping  
 46735 much of the environment control out of *posix\_spawn()* and *posix\_spawnp()* is appropriate design.

46736 The *posix\_spawn()* and *posix\_spawnp()* functions do not have all the power of *fork()/exec*. This is  
 46737 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to  
 46738 duplicate its functionality in a simple, fast function with no special hardware requirements. It is  
 46739 worth noting that *posix\_spawn()* and *posix\_spawnp()* are very similar to the process creation  
 46740 operations on many operating systems that are not UNIX systems.

**Requirements**

46741 The requirements for *posix\_spawn()* and *posix\_spawnp()* are:

- 46743 • They must be implementable without an MMU or unusual hardware.
- 46744 • They must be compatible with existing POSIX standards.

46745 Additional goals are:

- 46746 • They should be efficiently implementable.
- 46747 • They should be able to replace at least 50% of typical executions of *fork()*.

- 46748
- 46749
- A system with *posix\_spawn()* and *posix\_spawnp()* and without *fork()* should be useful, at least for realtime applications.
- 46750
- 46751
- A system with *fork()* and the *exec* family should be able to implement *posix\_spawn()* and *posix\_spawnp()* as library routines.

46752

### Two-Syntax

46753

46754

46755

46756

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix\_spawn\*()* functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix\_spawn\*()* functions.

46757

46758

The parameter list does not differ between *posix\_spawn()* and *posix\_spawnp()*; *posix\_spawnp()* interprets the second parameter more elaborately than *posix\_spawn()*.

46759

### Compatibility with POSIX.5 (Ada)

46760

46761

46762

46763

46764

46765

46766

46767

46768

46769

46770

46771

46772

46773

46774

The *Start\_Process* and *Start\_Process\_Search* procedures from the *POSIX\_Process\_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a manner similar to that of *posix\_spawn()* and *posix\_spawnp()*. Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix\_spawn()* and *posix\_spawnp()* to a subset of the capabilities of *Start\_Process* and *Start\_Process\_Search*; certain non-default capabilities were not supported. However, based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix\_spawn()* and *posix\_spawnp()* should be sufficiently powerful to implement *Start\_Process* and *Start\_Process\_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix\_spawn()* and *posix\_spawnp()* that are not provided by *Start\_Process* and *Start\_Process\_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

46775

46776

46777

46778

46779

46780

For the Ada language binding for *Start\_Process* to be implemented with *posix\_spawn()*, that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix\_spawn()* whenever the caller of *Start\_Process* allowed these arguments to default, since *posix\_spawn()* does not provide such defaults. The ability of *Start\_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix\_spawn()*.

46781

### Process Group

46782

46783

46784

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

**posix\_spawn()**46785 **Threads**

46786 Without the *posix\_spawn()* and *posix\_spawnnp()* functions, systems without address translation  
 46787 can still use threads to give an abstraction of concurrency. In many cases, thread creation  
 46788 suffices, but it is not always a good substitute. The *posix\_spawn()* and *posix\_spawnnp()* functions  
 46789 are considerably “heavier” than thread creation. Processes have several important attributes that  
 46790 threads do not. Even without address translation, a process may have base-and-bound memory  
 46791 protection. Each process has a process environment including security attributes and file  
 46792 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-  
 46793 uniform-memory-architecture multi-processors better than threads, and they are more  
 46794 convenient to use for activities that are not closely linked.

46795 The *posix\_spawn()* and *posix\_spawnnp()* functions may not bring support for multiple processes to  
 46796 every configuration. Process creation is not the only piece of operating system support required  
 46797 to support multiple processes. The total cost of support for multiple processes may be quite high  
 46798 in some circumstances. Existing practice shows that support for multiple processes is  
 46799 uncommon and threads are common among “tiny kernels”. There should, therefore, probably  
 46800 continue to be AEPs for operating systems with only one process.

46801 **Asynchronous Error Notification**

46802 A library implementation of *posix\_spawn()* or *posix\_spawnnp()* may not be able to detect all  
 46803 possible errors before it forks the child process. POSIX.1-2008 provides for an error indication  
 46804 returned from a child process which could not successfully complete the spawn operation via a  
 46805 special exit status which may be detected using the status value returned by *wait()*, *waitid()*, and  
 46806 *waitpid()*.

46807 The *stat\_val* interface and the macros used to interpret it are not well suited to the purpose of  
 46808 returning API errors, but they are the only path available to a library implementation. Thus, an  
 46809 implementation may cause the child process to exit with exit status 127 for any error detected  
 46810 during the spawn process after the *posix\_spawn()* or *posix\_spawnnp()* function has successfully  
 46811 returned.

46812 The standard developers had proposed using two additional macros to interpret *stat\_val*. The  
 46813 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because  
 46814 of an error detected during the *posix\_spawn()* or *posix\_spawnnp()* operations rather than during  
 46815 actual execution of the child process image; the second, WSPAWNERRNO, would have  
 46816 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group  
 46817 strongly opposed this because it would make a library implementation of *posix\_spawn()* or  
 46818 *posix\_spawnnp()* dependent on kernel modifications to *waitpid()* to be able to embed special  
 46819 information in *stat\_val* to indicate a spawn failure.

46820 The 8 bits of child process exit status that are guaranteed by POSIX.1-2008 to be accessible to the  
 46821 waiting parent process are insufficient to disambiguate a spawn error from any other kind of  
 46822 error that may be returned by an arbitrary process image. No other bits of the exit status are  
 46823 required to be visible in *stat\_val*, so these macros could not be strictly implemented at the library  
 46824 level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this  
 46825 value by *system()* and *popen()* to signal failures in these operations that occur after the function  
 46826 has returned but before a shell is able to execute. The exit status of 127 does not uniquely  
 46827 identify this class of error, nor does it provide any detailed information on the nature of the  
 46828 failure. Note that a kernel implementation of *posix\_spawn()* or *posix\_spawnnp()* is permitted (and  
 46829 encouraged) to return any possible error as the function value, thus providing more detailed  
 46830 failure information to the parent process.

46831 Thus, no special macros are available to isolate asynchronous *posix\_spawn()* or *posix\_spawnnp()*

46832 errors. Instead, errors detected by the *posix\_spawn()* or *posix\_spawnnp()* operations in the context  
 46833 of the child process before the new process image executes are reported by setting the child's exit  
 46834 status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the  
 46835 *stat\_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that  
 46836 other status values with which the child process image may exit (before the parent can  
 46837 conclusively determine that the child process image has begun execution) are distinct from exit  
 46838 status 127.

#### 46839 FUTURE DIRECTIONS

46840 None.

#### 46841 SEE ALSO

46842 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fstatat()*, *kill()*, *open()*,  
 46843 *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
 46844 *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
 46845 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
 46846 *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*, *sched\_setparam()*,  
 46847 *sched\_setscheduler()*, *setpgid()*, *setuid()*, *times()*, *wait()*, *waitid()*

46848 XBD Chapter 8 (on page 173), [<spawn.h>](#)

#### 46849 CHANGE HISTORY

46850 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46851 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are  
 46852 changed as well as the signal mask in step 2.

46853 IEEE PASC Interpretation 1003.1 #132 is applied.

#### 46854 Issue 7

46855 Functionality relating to the Threads option is moved to the Base.

**posix\_spawn\_file\_actions\_addclose()**

System Interfaces

46856 **NAME**

46857 **posix\_spawn\_file\_actions\_addclose**, **posix\_spawn\_file\_actions\_addopen** — add close or open  
 46858 action to spawn file actions object (**ADVANCED REALTIME**)

46859 **SYNOPSIS**

```
46860 SPN #include <spawn.h>
46861
46862 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
46863 *file_actions, int fildes);
46864 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
46865 *restrict file_actions, int fildes,
46866 const char *restrict path, int oflag, mode_t mode);
```

46866 **DESCRIPTION**

46867 These functions shall add or delete a close or open action to a spawn file actions object.

46868 A spawn file actions object is of type **posix\_spawn\_file\_actions\_t** (defined in **<spawn.h>**) and is  
 46869 used to specify a series of actions to be performed by a *posix\_spawn()* or *posix\_spawnp()*  
 46870 operation in order to arrive at the set of open file descriptors for the child process given the set  
 46871 of open file descriptors of the parent. POSIX.1-2008 does not define comparison or assignment  
 46872 operators for the type **posix\_spawn\_file\_actions\_t**.

46873 A spawn file actions object, when passed to *posix\_spawn()* or *posix\_spawnp()*, shall specify how  
 46874 the set of open file descriptors in the calling process is transformed into a set of potentially open  
 46875 file descriptors for the spawned process. This transformation shall be as if the specified sequence  
 46876 of actions was performed exactly once, in the context of the spawned process (prior to execution  
 46877 of the new process image), in the order in which the actions were added to the object;  
 46878 additionally, when the new process image is executed, any file descriptor (from this new set)  
 46879 which has its FD\_CLOEXEC flag set shall be closed (see *posix\_spawn()*).

46880 The *posix\_spawn\_file\_actions\_addclose()* function shall add a *close* action to the object referenced  
 46881 by *file\_actions* that shall cause the file descriptor *fildes* to be closed (as if *close(fildes)* had been  
 46882 called) when a new process is spawned using this file actions object.

46883 The *posix\_spawn\_file\_actions\_addopen()* function shall add an *open* action to the object referenced  
 46884 by *file\_actions* that shall cause the file named by *path* to be opened (as if *open(path, oflag, mode)*  
 46885 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a  
 46886 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,  
 46887 it shall be closed before the new file is opened.

46888 The string described by *path* shall be copied by the *posix\_spawn\_file\_actions\_addopen()* function.

46889 **RETURN VALUE**

46890 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 46891 be returned to indicate the error.

46892 **ERRORS**

46893 These functions shall fail if:

46894 [EBADF] The value specified by *fildes* is negative or greater than or equal to  
 46895 {OPEN\_MAX}.

46896 These functions may fail if:

46897 [EINVAL] The value specified by *file\_actions* is invalid.

46898 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

46899 It shall not be considered an error for the *fildev* argument passed to these functions to specify a  
 46900 file descriptor for which the specified operation could not be performed at the time of the call.  
 46901 Any such error will be detected when the associated file actions object is later used during a  
 46902 *posix\_spawn()* or *posix\_spawnp()* operation.

#### 46903 EXAMPLES

46904 None.

#### 46905 APPLICATION USAGE

46906 These functions are part of the Spawn option and need not be provided on all implementations.

#### 46907 RATIONALE

46908 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,  
 46909 and *open()* operations to be used by *posix\_spawn()* or *posix\_spawnp()* to arrive at the set of open  
 46910 file descriptors inherited by the spawned process from the set of open file descriptors in the  
 46911 parent at the time of the *posix\_spawn()* or *posix\_spawnp()* call. It had been suggested that the  
 46912 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files  
 46913 which need to be opened for use by the spawned process can be handled either by having the  
 46914 calling process open them before the *posix\_spawn()* or *posix\_spawnp()* call (and close them after),  
 46915 or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The  
 46916 standard developers recommend that applications use one of these two methods when practical,  
 46917 since detailed error status on a failed open operation is always available to the application this  
 46918 way. However, the standard developers feel that allowing a spawn file actions object to specify  
 46919 open operations is still appropriate because:

- 46920 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 46921 2. It supports the I/O redirection paradigm commonly employed by POSIX programs  
 46922 designed to be invoked from a shell. When such a program is the child process, it may not  
 46923 be designed to open files on its own.
- 46924 3. It allows file opens that might otherwise fail or violate file ownership/access rights if  
 46925 executed by the parent process.

46926 Regarding 2. above, note that the spawn open file action provides to *posix\_spawn()* and  
 46927 *posix\_spawnp()* the same capability that the shell redirection operators provide to *system()*, only  
 46928 without the intervening execution of a shell; for example:

```
46929 system ("myprog <file1 3<file2");
```

46930 Regarding 3. above, note that if the calling process needs to open one or more files for access by  
 46931 the spawned process, but has insufficient spare file descriptors, then the open action is necessary  
 46932 to allow the *open()* to occur in the context of the child process after other file descriptors have  
 46933 been closed (that must remain open in the parent).

46934 Additionally, if a parent is executed from a file having a "set-user-id" mode bit set and the  
 46935 POSIX\_SPAWN\_RESETIDS flag is set in the spawn attributes, a file created within the parent  
 46936 process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file  
 46937 created via an *open()* action during *posix\_spawn()* or *posix\_spawnp()* will have the parent's real  
 46938 ID as its owner; and an open by the parent process may successfully open a file to which the real  
 46939 user should not have access or fail to open a file to which the real user should have access.

**posix\_spawn\_file\_actions\_addclose()**

System Interfaces

46940 **File Descriptor Mapping**

46941 The standard developers had originally proposed using an array which specified the mapping of  
46942 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is  
46943 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix\_spawn()*  
46944 or *posix\_spawnp()* without provision for one or more spare file descriptor entries (which simply  
46945 may not be available). Such an array requires that an implementation develop a complex  
46946 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor  
46947 at the wrong time.

46948 It was noted by a member of the Ada Language Bindings working group that the approved Ada  
46949 Language *Start\_Process* family of POSIX process primitives use a caller-specified set of file  
46950 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very  
46951 flexible way, yet no such problems exist because the burden of determining how to achieve the  
46952 final file descriptor mapping is completely on the application. Furthermore, although the file  
46953 actions interface appears frightening at first glance, it is actually quite simple to implement in  
46954 either a library or the kernel.

46955 **FUTURE DIRECTIONS**

46956 None.

46957 **SEE ALSO**

46958 *close()*, *dup()*, *open()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_adddup2()*,  
46959 *posix\_spawn\_file\_actions\_destroy()*

46960 XBD <[spawn.h](#)>46961 **CHANGE HISTORY**

46962 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46963 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the  
46964 string pointed to by *path* is copied by the *posix\_spawn\_file\_actions\_addopen()* function.

46965 **NAME**

46966 `posix_spawn_file_actions_adddup2` — add `dup2` action to spawn file actions object  
 46967 (**ADVANCED REALTIME**)

46968 **SYNOPSIS**

```
46969 SPN #include <spawn.h>
46970 int posix_spawn_file_actions_adddup2 (posix_spawn_file_actions_t
46971 *file_actions, int fildes, int newfildes);
```

46972 **DESCRIPTION**

46973 The `posix_spawn_file_actions_adddup2()` function shall add a `dup2()` action to the object  
 46974 referenced by `file_actions` that shall cause the file descriptor `fildes` to be duplicated as `newfildes` (as  
 46975 if `dup2(fildes, newfildes)` had been called) when a new process is spawned using this file actions  
 46976 object.

46977 A spawn file actions object is as defined in `posix_spawn_file_actions_addclose()`.

46978 **RETURN VALUE**

46979 Upon successful completion, the `posix_spawn_file_actions_adddup2()` function shall return zero;  
 46980 otherwise, an error number shall be returned to indicate the error.

46981 **ERRORS**

46982 The `posix_spawn_file_actions_adddup2()` function shall fail if:

- 46983 [EBADF] The value specified by `fildes` or `newfildes` is negative or greater than or equal to  
 46984 {OPEN\_MAX}.
- 46985 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

46986 The `posix_spawn_file_actions_adddup2()` function may fail if:

- 46987 [EINVAL] The value specified by `file_actions` is invalid.

46988 It shall not be considered an error for the `fildes` argument passed to the  
 46989 `posix_spawn_file_actions_adddup2()` function to specify a file descriptor for which the specified  
 46990 operation could not be performed at the time of the call. Any such error will be detected when  
 46991 the associated file actions object is later used during a `posix_spawn()` or `posix_spawnnp()`  
 46992 operation.

46993 **EXAMPLES**

46994 None.

46995 **APPLICATION USAGE**

46996 The `posix_spawn_file_actions_adddup2()` function is part of the Spawn option and need not be  
 46997 provided on all implementations.

46998 **RATIONALE**

46999 Refer to the RATIONALE in `posix_spawn_file_actions_addclose()`.

47000 **FUTURE DIRECTIONS**

47001 None.

47002 **SEE ALSO**

47003 `dup()`, `posix_spawn()`, `posix_spawn_file_actions_addclose()`, `posix_spawn_file_actions_destroy()`

47004 XBD `<spawn.h>`

## posix\_spawn\_file\_actions\_adddup2()

System Interfaces

### 47005 CHANGE HISTORY

- 47006 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 47007 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the *newfildes* argument in addition to *fildes*.
- 47008

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47009 **NAME**

47010 `posix_spawn_file_actions_addopen` — add open action to spawn file actions object  
47011 **(ADVANCED REALTIME)**

47012 **SYNOPSIS**

```
47013 SPN #include <spawn.h>  
47014 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t  
47015 *restrict file_actions, int fildes,  
47016 const char *restrict path, int oflag, mode_t mode);
```

47017 **DESCRIPTION**

47018 Refer to [posix\\_spawn\\_file\\_actions\\_addclose\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawn\_file\_actions\_destroy()**

System Interfaces

47019 **NAME**

47020 `posix_spawn_file_actions_destroy`, `posix_spawn_file_actions_init` — destroy and initialize  
 47021 spawn file actions object (**ADVANCED REALTIME**)

47022 **SYNOPSIS**

```
47023 SPN      #include <spawn.h>
47024
47024      int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
47025          *file_actions);
47026
47026      int posix_spawn_file_actions_init(posix_spawn_file_actions_t
47027          *file_actions);
```

47028 **DESCRIPTION**

47029 The `posix_spawn_file_actions_destroy()` function shall destroy the object referenced by `file_actions`;  
 47030 the object becomes, in effect, uninitialized. An implementation may cause  
 47031 `posix_spawn_file_actions_destroy()` to set the object referenced by `file_actions` to an invalid value. A  
 47032 destroyed spawn file actions object can be reinitialized using `posix_spawn_file_actions_init()`; the  
 47033 results of otherwise referencing the object after it has been destroyed are undefined.

47034 The `posix_spawn_file_actions_init()` function shall initialize the object referenced by `file_actions` to  
 47035 contain no file actions for `posix_spawn()` or `posix_spawnnp()` to perform.

47036 A spawn file actions object is as defined in `posix_spawn_file_actions_addclose()`.

47037 The effect of initializing an already initialized spawn file actions object is undefined.

47038 **RETURN VALUE**

47039 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 47040 be returned to indicate the error.

47041 **ERRORS**

47042 The `posix_spawn_file_actions_init()` function shall fail if:

47043 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

47044 The `posix_spawn_file_actions_destroy()` function may fail if:

47045 [EINVAL] The value specified by `file_actions` is invalid.

47046 **EXAMPLES**

47047 None.

47048 **APPLICATION USAGE**

47049 These functions are part of the Spawn option and need not be provided on all implementations.

47050 **RATIONALE**

47051 Refer to the RATIONALE in `posix_spawn_file_actions_addclose()`.

47052 **FUTURE DIRECTIONS**

47053 None.

47054 **SEE ALSO**

47055 `posix_spawn()`

47056 XBD `<spawn.h>`

**CHANGE HISTORY**

- 47057 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 47058
- 47059 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_destroy()**47060 **NAME**

47061 `posix_spawnattr_destroy`, `posix_spawnattr_init` — destroy and initialize spawn attributes object  
 47062 (ADVANCED REALTIME)

47063 **SYNOPSIS**

```
47064 SPN #include <spawn.h>
47065
47065 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
47066 int posix_spawnattr_init(posix_spawnattr_t *attr);
```

47067 **DESCRIPTION**

47068 The `posix_spawnattr_destroy()` function shall destroy a spawn attributes object. A destroyed `attr`  
 47069 attributes object can be reinitialized using `posix_spawnattr_init()`; the results of otherwise  
 47070 referencing the object after it has been destroyed are undefined. An implementation may cause  
 47071 `posix_spawnattr_destroy()` to set the object referenced by `attr` to an invalid value.

47072 The `posix_spawnattr_init()` function shall initialize a spawn attributes object `attr` with the default  
 47073 value for all of the individual attributes used by the implementation. Results are undefined if  
 47074 `posix_spawnattr_init()` is called specifying an already initialized `attr` attributes object.

47075 A spawn attributes object is of type `posix_spawnattr_t` (defined in `<spawn.h>`) and is used to  
 47076 specify the inheritance of process attributes across a spawn operation. POSIX.1-2008 does not  
 47077 define comparison or assignment operators for the type `posix_spawnattr_t`.

47078 Each implementation shall document the individual attributes it uses and their default values  
 47079 unless these values are defined by POSIX.1-2008. Attributes not defined by POSIX.1-2008, their  
 47080 default values, and the names of the associated functions to get and set those attribute values are  
 47081 implementation-defined.

47082 The resulting spawn attributes object (possibly modified by setting individual attribute values),  
 47083 is used to modify the behavior of `posix_spawn()` or `posix_spawnnp()`. After a spawn attributes  
 47084 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnnp()`, any  
 47085 function affecting the attributes object (including destruction) shall not affect any process that  
 47086 has been spawned in this way.

47087 **RETURN VALUE**

47088 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return  
 47089 zero; otherwise, an error number shall be returned to indicate the error.

47090 **ERRORS**

47091 The `posix_spawnattr_init()` function shall fail if:

47092 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

47093 The `posix_spawnattr_destroy()` function may fail if:

47094 [EINVAL] The value specified by `attr` is invalid.

47095 **EXAMPLES**

47096 None.

47097 **APPLICATION USAGE**

47098 These functions are part of the Spawn option and need not be provided on all implementations.

47099 **RATIONALE**

47100 The original spawn interface proposed in POSIX.1-2008 defined the attributes that specify the  
 47101 inheritance of process attributes across a spawn operation as a structure. In order to be able to  
 47102 separate optional individual attributes under their appropriate options (that is, the `spawn-`  
 47103 `schedparam` and `spawn-schedpolicy` attributes depending upon the Process Scheduling option), and

47104 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface  
47105 has been changed to an opaque data type. This interface now consists of the type  
47106 **posix\_spawnattr\_t**, representing a spawn attributes object, together with associated functions to  
47107 initialize or destroy the attributes object, and to set or get each individual attribute. Although the  
47108 new object-oriented interface is more verbose than the original structure, it is simple to use,  
47109 more extensible, and easy to implement.

**47110 FUTURE DIRECTIONS**

47111 None.

**47112 SEE ALSO**

47113 *posix\_spawn()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
47114 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
47115 *posix\_spawnattr\_getsigmask()*

47116 XBD <spawn.h>

**47117 CHANGE HISTORY**

47118 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

47119 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already  
47120 initialized spawn attributes option is undefined.

**posix\_spawnattr\_getflags()**

System Interfaces

47121 **NAME**

47122 `posix_spawnattr_getflags`, `posix_spawnattr_setflags` — get and set the spawn-flags attribute of a  
 47123 spawn attributes object (**ADVANCED REALTIME**)

47124 **SYNOPSIS**

```
47125 SPN #include <spawn.h>
47126
47126 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
47127 short *restrict flags);
47128 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

47129 **DESCRIPTION**

47130 The `posix_spawnattr_getflags()` function shall obtain the value of the *spawn-flags* attribute from the  
 47131 attributes object referenced by *attr*.

47132 The `posix_spawnattr_setflags()` function shall set the *spawn-flags* attribute in an initialized  
 47133 attributes object referenced by *attr*.

47134 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the  
 47135 new process image when invoking `posix_spawn()` or `posix_spawnwp()`. It is the bitwise-inclusive  
 47136 OR of zero or more of the following flags:

```
47137 POSIX_SPAWN_RESETIDS
47138 POSIX_SPAWN_SETPGROUP
47139 POSIX_SPAWN_SETSIGDEF
47140 POSIX_SPAWN_SETSIGMASK
47141 PS POSIX_SPAWN_SETSCHEDPARAM
47142 POSIX_SPAWN_SETSCHEDULER
```

47143 These flags are defined in `<spawn.h>`. The default value of this attribute shall be as if no flags  
 47144 were set.

47145 **RETURN VALUE**

47146 Upon successful completion, `posix_spawnattr_getflags()` shall return zero and store the value of  
 47147 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an  
 47148 error number shall be returned to indicate the error.

47149 Upon successful completion, `posix_spawnattr_setflags()` shall return zero; otherwise, an error  
 47150 number shall be returned to indicate the error.

47151 **ERRORS**

47152 These functions may fail if:

47153 [EINVAL] The value specified by *attr* is invalid.

47154 The `posix_spawnattr_setflags()` function may fail if:

47155 [EINVAL] The value of the attribute being set is not valid.

**47156 EXAMPLES**

47157 None.

**47158 APPLICATION USAGE**

47159 These functions are part of the Spawn option and need not be provided on all implementations.

**47160 RATIONALE**

47161 None.

**47162 FUTURE DIRECTIONS**

47163 None.

**47164 SEE ALSO**

47165 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
47166 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
47167 *posix\_spawnattr\_getsigmask()*

47168 XBD <spawn.h>

**47169 CHANGE HISTORY**

47170 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_getpgroup()**

System Interfaces

47171 **NAME**

47172 `posix_spawnattr_getpgroup`, `posix_spawnattr_setpgroup` — get and set the spawn-pgroup  
 47173 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47174 **SYNOPSIS**

```
47175 SPN #include <spawn.h>
47176
47176 int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
47177 pid_t *restrict pgroup);
47178 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

47179 **DESCRIPTION**

47180 The `posix_spawnattr_getpgroup()` function shall obtain the value of the `spawn-pgroup` attribute  
 47181 from the attributes object referenced by `attr`.

47182 The `posix_spawnattr_setpgroup()` function shall set the `spawn-pgroup` attribute in an initialized  
 47183 attributes object referenced by `attr`.

47184 The `spawn-pgroup` attribute represents the process group to be joined by the new process image  
 47185 in a spawn operation (if `POSIX_SPAWN_SETPGROUP` is set in the `spawn-flags` attribute). The  
 47186 default value of this attribute shall be zero.

47187 **RETURN VALUE**

47188 Upon successful completion, `posix_spawnattr_getpgroup()` shall return zero and store the value of  
 47189 the `spawn-pgroup` attribute of `attr` into the object referenced by the `pgroup` parameter; otherwise,  
 47190 an error number shall be returned to indicate the error.

47191 Upon successful completion, `posix_spawnattr_setpgroup()` shall return zero; otherwise, an error  
 47192 number shall be returned to indicate the error.

47193 **ERRORS**

47194 These functions may fail if:

47195 [EINVAL] The value specified by `attr` is invalid.

47196 The `posix_spawnattr_setpgroup()` function may fail if:

47197 [EINVAL] The value of the attribute being set is not valid.

47198 **EXAMPLES**

47199 None.

47200 **APPLICATION USAGE**

47201 These functions are part of the Spawn option and need not be provided on all implementations.

47202 **RATIONALE**

47203 None.

47204 **FUTURE DIRECTIONS**

47205 None.

47206 **SEE ALSO**

47207 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,  
 47208 `posix_spawnattr_getflags()`, `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`,  
 47209 `posix_spawnattr_getsigmask()`

47210 XBD `<spawn.h>`

47211 **CHANGE HISTORY**

47212 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_getschedparam()**

System Interfaces

47213 **NAME**

47214 `posix_spawnattr_getschedparam`, `posix_spawnattr_setschedparam` — get and set the `spawn-schedparam`  
 47215 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47216 **SYNOPSIS**

```
47217 SPN PS #include <spawn.h>
47218 #include <sched.h>

47219 int posix_spawnattr_getschedparam(const posix_spawnattr_t
47220     *restrict attr, struct sched_param *restrict schedparam);
47221 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
47222     const struct sched_param *restrict schedparam);
```

47223 **DESCRIPTION**

47224 The `posix_spawnattr_getschedparam()` function shall obtain the value of the `spawn-schedparam`  
 47225 attribute from the attributes object referenced by `attr`.

47226 The `posix_spawnattr_setschedparam()` function shall set the `spawn-schedparam` attribute in an  
 47227 initialized attributes object referenced by `attr`.

47228 The `spawn-schedparam` attribute represents the scheduling parameters to be assigned to the new  
 47229 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` or  
 47230 `POSIX_SPAWN_SETSCHEDPARAM` is set in the `spawn-flags` attribute). The default value of this  
 47231 attribute is unspecified.

47232 **RETURN VALUE**

47233 Upon successful completion, `posix_spawnattr_getschedparam()` shall return zero and store the  
 47234 value of the `spawn-schedparam` attribute of `attr` into the object referenced by the `schedparam`  
 47235 parameter; otherwise, an error number shall be returned to indicate the error.

47236 Upon successful completion, `posix_spawnattr_setschedparam()` shall return zero; otherwise, an  
 47237 error number shall be returned to indicate the error.

47238 **ERRORS**

47239 These functions may fail if:

47240 [EINVAL] The value specified by `attr` is invalid.

47241 The `posix_spawnattr_setschedparam()` function may fail if:

47242 [EINVAL] The value of the attribute being set is not valid.

47243 **EXAMPLES**

47244 None.

47245 **APPLICATION USAGE**

47246 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 47247 on all implementations.

47248 **RATIONALE**

47249 None.

47250 **FUTURE DIRECTIONS**

47251 None.

47252 **SEE ALSO**

47253 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,  
 47254 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedpolicy()`,  
 47255 `posix_spawnattr_getsigmask()`

47256 XBD <sched.h>, <spawn.h>

47257 **CHANGE HISTORY**

47258 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_getschedpolicy()**

System Interfaces

47259 **NAME**

47260 `posix_spawnattr_getschedpolicy`, `posix_spawnattr_setschedpolicy` — get and set the spawn-  
 47261 `schedpolicy` attribute of a spawn attributes object (**ADVANCED REALTIME**)

47262 **SYNOPSIS**

```
47263 SPN PS #include <spawn.h>
47264 #include <sched.h>

47265 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
47266     *restrict attr, int *restrict schedpolicy);
47267 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
47268     int schedpolicy);
```

47269 **DESCRIPTION**

47270 The `posix_spawnattr_getschedpolicy()` function shall obtain the value of the `spawn-schedpolicy`  
 47271 attribute from the attributes object referenced by `attr`.

47272 The `posix_spawnattr_setschedpolicy()` function shall set the `spawn-schedpolicy` attribute in an  
 47273 initialized attributes object referenced by `attr`.

47274 The `spawn-schedpolicy` attribute represents the scheduling policy to be assigned to the new  
 47275 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` is set in the `spawn-`  
 47276 `flags` attribute). The default value of this attribute is unspecified.

47277 **RETURN VALUE**

47278 Upon successful completion, `posix_spawnattr_getschedpolicy()` shall return zero and store the  
 47279 value of the `spawn-schedpolicy` attribute of `attr` into the object referenced by the `schedpolicy`  
 47280 parameter; otherwise, an error number shall be returned to indicate the error.

47281 Upon successful completion, `posix_spawnattr_setschedpolicy()` shall return zero; otherwise, an  
 47282 error number shall be returned to indicate the error.

47283 **ERRORS**

47284 These functions may fail if:

47285 [EINVAL] The value specified by `attr` is invalid.

47286 The `posix_spawnattr_setschedpolicy()` function may fail if:

47287 [EINVAL] The value of the attribute being set is not valid.

47288 **EXAMPLES**

47289 None.

47290 **APPLICATION USAGE**

47291 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 47292 on all implementations.

47293 **RATIONALE**

47294 None.

47295 **FUTURE DIRECTIONS**

47296 None.

47297 **SEE ALSO**

47298 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,  
 47299 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`,  
 47300 `posix_spawnattr_getsigmask()`

47301 XBD `<sched.h>`, `<spawn.h>`

47302 **CHANGE HISTORY**

47303 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_getsigdefault()**47304 **NAME**

47305 `posix_spawnattr_getsigdefault`, `posix_spawnattr_setsigdefault` — get and set the `spawn-sigdefault`  
 47306 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47307 **SYNOPSIS**

```
47308 #include <signal.h>
47309 #include <spawn.h>

47310 int posix_spawnattr_getsigdefault(const posix_spawnattr_t
47311     *restrict attr, sigset_t *restrict sigdefault);
47312 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
47313     const sigset_t *restrict sigdefault);
```

47314 **DESCRIPTION**

47315 The `posix_spawnattr_getsigdefault()` function shall obtain the value of the `spawn-sigdefault`  
 47316 attribute from the attributes object referenced by `attr`.

47317 The `posix_spawnattr_setsigdefault()` function shall set the `spawn-sigdefault` attribute in an  
 47318 initialized attributes object referenced by `attr`.

47319 The `spawn-sigdefault` attribute represents the set of signals to be forced to default signal handling  
 47320 in the new process image (if `POSIX_SPAWN_SETSIGDEF` is set in the `spawn-flags` attribute) by a  
 47321 spawn operation. The default value of this attribute shall be an empty signal set.

47322 **RETURN VALUE**

47323 Upon successful completion, `posix_spawnattr_getsigdefault()` shall return zero and store the value  
 47324 of the `spawn-sigdefault` attribute of `attr` into the object referenced by the `sigdefault` parameter;  
 47325 otherwise, an error number shall be returned to indicate the error.

47326 Upon successful completion, `posix_spawnattr_setsigdefault()` shall return zero; otherwise, an error  
 47327 number shall be returned to indicate the error.

47328 **ERRORS**

47329 These functions may fail if:

47330 [EINVAL] The value specified by `attr` is invalid.

47331 The `posix_spawnattr_setsigdefault()` function may fail if:

47332 [EINVAL] The value of the attribute being set is not valid.

47333 **EXAMPLES**

47334 None.

47335 **APPLICATION USAGE**

47336 These functions are part of the Spawn option and need not be provided on all implementations.

47337 **RATIONALE**

47338 None.

47339 **FUTURE DIRECTIONS**

47340 None.

47341 **SEE ALSO**

47342 [posix\\_spawn\(\)](#), [posix\\_spawnattr\\_destroy\(\)](#), [posix\\_spawnattr\\_getflags\(\)](#), [posix\\_spawnattr\\_getpgroup\(\)](#),  
 47343 [posix\\_spawnattr\\_getschedparam\(\)](#), [posix\\_spawnattr\\_getschedpolicy\(\)](#), [posix\\_spawnattr\\_getsigmask\(\)](#)

47344 XBD [<signal.h>](#), [<spawn.h>](#)

47345 **CHANGE HISTORY**

47346 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_getsigmask()**47347 **NAME**

47348 `posix_spawnattr_getsigmask`, `posix_spawnattr_setsigmask` — get and set the `spawn-sigmask`  
 47349 attribute of a `spawn` attributes object (**ADVANCED REALTIME**)

47350 **SYNOPSIS**

```
47351 #include <signal.h>
47352 #include <spawn.h>

47353 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
47354                               sigset_t *restrict sigmask);
47355 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
47356                               const sigset_t *restrict sigmask);
```

47357 **DESCRIPTION**

47358 The `posix_spawnattr_getsigmask()` function shall obtain the value of the `spawn-sigmask` attribute  
 47359 from the attributes object referenced by `attr`.

47360 The `posix_spawnattr_setsigmask()` function shall set the `spawn-sigmask` attribute in an initialized  
 47361 attributes object referenced by `attr`.

47362 The `spawn-sigmask` attribute represents the signal mask in effect in the new process image of a  
 47363 `spawn` operation (if `POSIX_SPAWN_SETSIGMASK` is set in the `spawn-flags` attribute). The  
 47364 default value of this attribute is unspecified.

47365 **RETURN VALUE**

47366 Upon successful completion, `posix_spawnattr_getsigmask()` shall return zero and store the value  
 47367 of the `spawn-sigmask` attribute of `attr` into the object referenced by the `sigmask` parameter;  
 47368 otherwise, an error number shall be returned to indicate the error.

47369 Upon successful completion, `posix_spawnattr_setsigmask()` shall return zero; otherwise, an error  
 47370 number shall be returned to indicate the error.

47371 **ERRORS**

47372 These functions may fail if:

47373 [EINVAL] The value specified by `attr` is invalid.

47374 The `posix_spawnattr_setsigmask()` function may fail if:

47375 [EINVAL] The value of the attribute being set is not valid.

47376 **EXAMPLES**

47377 None.

47378 **APPLICATION USAGE**

47379 These functions are part of the `Spawn` option and need not be provided on all implementations.

47380 **RATIONALE**

47381 None.

47382 **FUTURE DIRECTIONS**

47383 None.

47384 **SEE ALSO**

47385 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,  
 47386 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`,  
 47387 `posix_spawnattr_getschedpolicy()`

47388 XBD `<signal.h>`, `<spawn.h>`

47389 **CHANGE HISTORY**

47390 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_init()**

System Interfaces

47391 **NAME**47392        posix\_spawnattr\_init — initialize the spawn attributes object (**ADVANCED REALTIME**)47393 **SYNOPSIS**

47394 SPN     #include &lt;spawn.h&gt;

47395        int posix\_spawnattr\_init(posix\_spawnattr\_t \*attr);

47396 **DESCRIPTION**47397        Refer to *posix\_spawnattr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47398 **NAME**

47399 `posix_spawnattr_setflags` — set the spawn-flags attribute of a spawn attributes object  
47400 (**ADVANCED REALTIME**)

47401 **SYNOPSIS**

```
47402 SPN #include <spawn.h>  
47403 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

47404 **DESCRIPTION**

47405 Refer to [posix\\_spawnattr\\_getflags\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_setpgroup()**

System Interfaces

47406 **NAME**

47407        `posix_spawnattr_setpgroup` — set the spawn-pgroup attribute of a spawn attributes object  
47408        (**ADVANCED REALTIME**)

47409 **SYNOPSIS**

```
47410 SPN    #include <spawn.h>  
47411        int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

47412 **DESCRIPTION**

47413        Refer to *posix\_spawnattr\_getpgroup()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47414 **NAME**

47415 `posix_spawnattr_setschedparam` — set the spawn-schedparam attribute of a spawn attributes  
47416 object (**ADVANCED REALTIME**)

47417 **SYNOPSIS**

```
47418 SPN PS #include <sched.h>  
47419 #include <spawn.h>  
  
47420 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
47421     const struct sched_param *restrict schedparam);
```

47422 **DESCRIPTION**

47423 Refer to [posix\\_spawnattr\\_getschedparam\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_setschedpolicy()**

System Interfaces

47424 **NAME**

47425 `posix_spawnattr_setschedpolicy` — set the spawn-schedpolicy attribute of a spawn attributes  
47426 object (**ADVANCED REALTIME**)

47427 **SYNOPSIS**

```
47428 SPN PS #include <sched.h>  
47429 #include <spawn.h>  
  
47430 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
47431 int schedpolicy);
```

47432 **DESCRIPTION**

47433 Refer to [posix\\_spawnattr\\_getschedpolicy\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47434 **NAME**

47435 `posix_spawnattr_setsigdefault` — set the spawn-sigdefault attribute of a spawn attributes object  
47436 (**ADVANCED REALTIME**)

47437 **SYNOPSIS**

```
47438 SPN #include <signal.h>  
47439 #include <spawn.h>  
  
47440 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
47441 const sigset_t *restrict sigdefault);
```

47442 **DESCRIPTION**

47443 Refer to [posix\\_spawnattr\\_getsigdefault\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_spawnattr\_setsigmask()**

System Interfaces

47444 **NAME**

47445 `posix_spawnattr_setsigmask` — set the spawn-sigmask attribute of a spawn attributes object  
47446 (**ADVANCED REALTIME**)

47447 **SYNOPSIS**

```
47448 SPN #include <signal.h>  
47449 #include <spawn.h>  
  
47450 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
47451 const sigset_t *restrict sigmask);
```

47452 **DESCRIPTION**

47453 Refer to [posix\\_spawnattr\\_getsigmask\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47454 **NAME**47455 `posix_spawn` — spawn a process (ADVANCED REALTIME)47456 **SYNOPSIS**

```
47457 SPN #include <spawn.h>
47458
47458 int posix_spawn(pid_t *restrict pid, const char *restrict file,
47459                const posix_spawn_file_actions_t *file_actions,
47460                const posix_spawnattr_t *restrict attrp,
47461                char *const argv[restrict], char *const envp[restrict]);
```

47462 **DESCRIPTION**47463 Refer to *posix\_spawn()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_destroy()**

System Interfaces

47464 **NAME**

47465 `posix_trace_attr_destroy`, `posix_trace_attr_init` — destroy and initialize the trace stream  
 47466 attributes object (**TRACING**)

47467 **SYNOPSIS**

```
47468 OB TRC #include <trace.h>
47469
47469 int posix_trace_attr_destroy(trace_attr_t *attr);
47470 int posix_trace_attr_init(trace_attr_t *attr);
```

47471 **DESCRIPTION**

47472 The `posix_trace_attr_destroy()` function shall destroy an initialized trace attributes object. A  
 47473 destroyed `attr` attributes object can be reinitialized using `posix_trace_attr_init()`; the results of  
 47474 otherwise referencing the object after it has been destroyed are undefined.

47475 The `posix_trace_attr_init()` function shall initialize a trace attributes object `attr` with the default  
 47476 value for all of the individual attributes used by a given implementation. The read-only  
 47477 `generation-version` and `clock-resolution` attributes of the newly initialized trace attributes object  
 47478 shall be set to their appropriate values (see [Section 2.11.1.2](#), on page 535).

47479 Results are undefined if `posix_trace_attr_init()` is called specifying an already initialized `attr`  
 47480 attributes object.

47481 Implementations may add extensions to the trace attributes object structure as permitted in XBD  
 47482 [Chapter 2](#) (on page 15).

47483 The resulting attributes object (possibly modified by setting individual attributes values), when  
 47484 used by `posix_trace_create()`, defines the attributes of the trace stream created. A single attributes  
 47485 object can be used in multiple calls to `posix_trace_create()`. After one or more trace streams have  
 47486 been created using an attributes object, any function affecting that attributes object, including  
 47487 destruction, shall not affect any trace stream previously created. An initialized attributes object  
 47488 also serves to receive the attributes of an existing trace stream or trace log when calling the  
 47489 `posix_trace_get_attr()` function.

47490 **RETURN VALUE**

47491 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 47492 return the corresponding error number.

47493 **ERRORS**

47494 The `posix_trace_attr_destroy()` function may fail if:

47495 [EINVAL] The value of `attr` is invalid.

47496 The `posix_trace_attr_init()` function shall fail if:

47497 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

47498 **EXAMPLES**

47499 None.

47500 **APPLICATION USAGE**

47501 None.

47502 **RATIONALE**

47503 None.

**47504 FUTURE DIRECTIONS**

47505       The *posix\_trace\_attr\_destroy()* and *posix\_trace\_attr\_init()* functions may be removed in a future  
47506       version.

**47507 SEE ALSO**

47508       *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*

47509       XBD <trace.h>

**47510 CHANGE HISTORY**

47511       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47512       IEEE PASC Interpretation 1003.1 #123 is applied.

**47513 Issue 7**

47514       The *posix\_trace\_attr\_destroy()* and *posix\_trace\_attr\_init()* functions are marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_getclockres()**

System Interfaces

47515 **NAME**

47516 `posix_trace_attr_getclockres`, `posix_trace_attr_getcreatetime`, `posix_trace_attr_getgenversion`,  
 47517 `posix_trace_attr_getname`, `posix_trace_attr_setname` — retrieve and set information about a  
 47518 trace stream (**TRACING**)

47519 **SYNOPSIS**

```
47520 OB TRC #include <time.h>
47521 #include <trace.h>

47522 int posix_trace_attr_getclockres(const trace_attr_t *attr,
47523     struct timespec *resolution);
47524 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
47525     struct timespec *createtime);

47526 #include <trace.h>

47527 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
47528     char *genversion);
47529 int posix_trace_attr_getname(const trace_attr_t *attr,
47530     char *tracename);
47531 int posix_trace_attr_setname(trace_attr_t *attr,
47532     const char *tracename);
```

47533 **DESCRIPTION**

47534 The `posix_trace_attr_getclockres()` function shall copy the clock resolution of the clock used to  
 47535 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the  
 47536 *attr* argument into the structure pointed to by the *resolution* argument.

47537 The `posix_trace_attr_getcreatetime()` function shall copy the trace stream creation time from the  
 47538 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure  
 47539 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of  
 47540 creation of the trace stream.

47541 The `posix_trace_attr_getgenversion()` function shall copy the string containing version information  
 47542 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into  
 47543 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of  
 47544 a character array which can store at least {TRACE\_NAME\_MAX} characters.

47545 The `posix_trace_attr_getname()` function shall copy the string containing the trace name from the  
 47546 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string  
 47547 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character  
 47548 array which can store at least {TRACE\_NAME\_MAX} characters.

47549 The `posix_trace_attr_setname()` function shall set the name in the *trace-name* attribute of the  
 47550 attributes object pointed to by the *attr* argument, using the trace name string supplied by the  
 47551 *tracename* argument. If the supplied string contains more than {TRACE\_NAME\_MAX}  
 47552 characters, the name copied into the *trace-name* attribute may be truncated to one less than the  
 47553 length of {TRACE\_NAME\_MAX} characters. The default value is a null string.

47554 **RETURN VALUE**

47555 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 47556 return the corresponding error number.

47557 If successful, the `posix_trace_attr_getclockres()` function stores the *clock-resolution* attribute value in  
 47558 the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

47559 If successful, the `posix_trace_attr_getcreatetime()` function stores the trace stream creation time in

47560 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

47561 If successful, the *posix\_trace\_attr\_getgenversion()* function stores the trace version information in  
47562 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

47563 If successful, the *posix\_trace\_attr\_getname()* function stores the trace name in the string pointed  
47564 to by *tracename*. Otherwise, the content of this string is unspecified.

#### 47565 **ERRORS**

47566 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
47567 and *posix\_trace\_attr\_getname()* functions may fail if:

47568 [EINVAL] The value specified by one of the arguments is invalid.

#### 47569 **EXAMPLES**

47570 None.

#### 47571 **APPLICATION USAGE**

47572 None.

#### 47573 **RATIONALE**

47574 None.

#### 47575 **FUTURE DIRECTIONS**

47576 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
47577 *posix\_trace\_attr\_getname()*, and *posix\_trace\_attr\_setname()* functions may be removed in a future  
47578 version.

#### 47579 **SEE ALSO**

47580 *posix\_trace\_attr\_destroy()*, *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*

47581 XBD [<time.h>](#), [<trace.h>](#)

#### 47582 **CHANGE HISTORY**

47583 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

#### 47584 **Issue 7**

47585 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
47586 *posix\_trace\_attr\_getname()*, and *posix\_trace\_attr\_setname()* functions are marked obsolescent.

**posix\_trace\_attr\_getinherited()**

System Interfaces

47587 **NAME**

47588 `posix_trace_attr_getinherited`, `posix_trace_attr_getlogfullpolicy`,  
 47589 `posix_trace_attr_getstreamfullpolicy`, `posix_trace_attr_setinherited`,  
 47590 `posix_trace_attr_setlogfullpolicy`, `posix_trace_attr_setstreamfullpolicy` — retrieve and set the  
 47591 behavior of a trace stream (**TRACING**)

47592 **SYNOPSIS**

```
47593 OB TRC #include <trace.h>
47594 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
47595 int *restrict inheritancepolicy);
47596 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
47597 int *restrict logpolicy);
47598 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
47599 attr, int *restrict streampolicy);
47600 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
47601 int inheritancepolicy);
47602 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
47603 int logpolicy);
47604 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
47605 int streampolicy);
```

47606 **DESCRIPTION**

47607 TRI The `posix_trace_attr_getinherited()` and `posix_trace_attr_setinherited()` functions, respectively, shall  
 47608 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the  
 47609 `fork()` and `spawn()` operations. The *inheritance* attribute of the attributes object pointed to by the  
 47610 *attr* argument shall be set to one of the following values defined by manifest constants in the  
 47611 **<trace.h>** header:

47612 **POSIX\_TRACE\_CLOSE\_FOR\_CHILD**

47613 After a `fork()` or `spawn()` operation, the child shall not be traced, and tracing of the parent  
 47614 shall continue.

47615 **POSIX\_TRACE\_INHERITED**

47616 After a `fork()` or `spawn()` operation, if the parent is being traced, its child shall be  
 47617 concurrently traced using the same trace stream.

47618 The default value for the *inheritance* attribute is **POSIX\_TRACE\_CLOSE\_FOR\_CHILD**.

47619 TRL The `posix_trace_attr_getlogfullpolicy()` and `posix_trace_attr_setlogfullpolicy()` functions,  
 47620 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the  
 47621 attributes object pointed to by the *attr* argument.

47622 The *log-full-policy* attribute shall be set to one of the following values defined by manifest  
 47623 constants in the **<trace.h>** header:

47624 **POSIX\_TRACE\_LOOP**

47625 The trace log shall loop until the associated trace stream is stopped. This policy means that  
 47626 when the trace log gets full, the file system shall reuse the resources allocated to the oldest  
 47627 trace events that were recorded. In this way, the trace log will always contain the most  
 47628 recent trace events flushed.

47629 **POSIX\_TRACE\_UNTIL\_FULL**

47630 The trace stream shall be flushed to the trace log until the trace log is full. This condition can  
 47631 be deduced from the *posix\_log\_full\_status* member status (see the **posix\_trace\_status\_info**  
 47632 structure defined in **<trace.h>**). The last recorded trace event shall be the  
 47633 **POSIX\_TRACE\_STOP** trace event.

- 47634 **POSIX\_TRACE\_APPEND**  
 47635 The associated trace stream shall be flushed to the trace log without log size limitation. If  
 47636 the application specifies **POSIX\_TRACE\_APPEND**, the implementation shall ignore the *log-*  
 47637 *max-size* attribute.
- 47638 The default value for the *log-full-policy* attribute is **POSIX\_TRACE\_LOOP**.
- 47639 The *posix\_trace\_attr\_getstreamfullpolicy()* and *posix\_trace\_attr\_setstreamfullpolicy()* functions  
 47640 respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute  
 47641 of the attributes object pointed to by the *attr* argument.
- 47642 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest  
 47643 constants in the **<trace.h>** header:
- 47644 **POSIX\_TRACE\_LOOP**  
 47645 The trace stream shall loop until explicitly stopped by the *posix\_trace\_stop()* function. This  
 47646 policy means that when the trace stream is full, the trace system shall reuse the resources  
 47647 allocated to the oldest trace events recorded. In this way, the trace stream will always  
 47648 contain the most recent trace events recorded.
- 47649 **POSIX\_TRACE\_UNTIL\_FULL**  
 47650 The trace stream will run until the trace stream resources are exhausted. Then the trace  
 47651 stream will stop. This condition can be deduced from *posix\_stream\_status* and  
 47652 *posix\_stream\_full\_status* (see the **posix\_trace\_status\_info** structure defined in **<trace.h>**).  
 47653 When this trace stream is read, a **POSIX\_TRACE\_STOP** trace event shall be reported after  
 47654 reporting the last recorded trace event. The trace system shall reuse the resources allocated  
 47655 to any trace events already reported—see the *posix\_trace\_getnext\_event()*,  
 47656 *posix\_trace\_trygetnext\_event()*, and *posix\_trace\_timedgetnext\_event()* functions—or already  
 47657 flushed for an active trace stream with log if the Trace Log option is supported; see the  
 47658 *posix\_trace\_flush()* function. The trace system shall restart the trace stream when it is empty  
 47659 and may restart it sooner. A **POSIX\_TRACE\_START** trace event shall be reported before  
 47660 reporting the next recorded trace event.
- 47661 **POSIX\_TRACE\_FLUSH**  
 47662 If the Trace Log option is supported, this policy is identical to the  
 47663 **POSIX\_TRACE\_UNTIL\_FULL** trace stream full policy except that the trace stream shall be  
 47664 flushed regularly as if *posix\_trace\_flush()* had been explicitly called. Defining this policy for  
 47665 an active trace stream without log shall be invalid.
- 47666 The default value for the *stream-full-policy* attribute shall be **POSIX\_TRACE\_LOOP** for an active  
 47667 trace stream without log.
- 47668 **POSIX\_TRACE\_FLUSH**  
 47669 If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be  
**POSIX\_TRACE\_FLUSH** for an active trace stream with log.
- 47670 **RETURN VALUE**  
 47671 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 47672 return the corresponding error number.
- 47673 **inheritance**  
 47674 If successful, the *posix\_trace\_attr\_getinherited()* function shall store the *inheritance* attribute value  
 in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.
- 47675 **log-policy**  
 47676 If successful, the *posix\_trace\_attr\_getlogfullpolicy()* function shall store the *log-full-policy* attribute  
 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.
- 47677 **stream-policy**  
 47678 If successful, the *posix\_trace\_attr\_getstreamfullpolicy()* function shall store the *stream-full-policy*  
 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is  
 47679 undefined.

**posix\_trace\_attr\_getinherited()**47680 **ERRORS**

47681 These functions may fail if:

47682 [EINVAL] The value specified by at least one of the arguments is invalid.

47683 **EXAMPLES**

47684 None.

47685 **APPLICATION USAGE**

47686 None.

47687 **RATIONALE**

47688 None.

47689 **FUTURE DIRECTIONS**

47690 The following functions:

47691 *posix\_trace\_attr\_getinherited()*  
 47692 *posix\_trace\_attr\_getlogfullpolicy()*  
 47693 *posix\_trace\_attr\_getstreamfullpolicy()*  
 47694 *posix\_trace\_attr\_setinherited()*  
 47695 *posix\_trace\_attr\_setlogfullpolicy()*  
 47696 *posix\_trace\_attr\_setstreamfullpolicy()*

47697 may be removed in a future version.

47698 **SEE ALSO**

47699 *fork()*, *posix\_trace\_attr\_destroy()*, *posix\_trace\_create()*, *posix\_trace\_get\_attr()*,  
 47700 *posix\_trace\_getnext\_event()*, *posix\_trace\_start()*

47701 XBD &lt;trace.h&gt;

47702 **CHANGE HISTORY**

47703 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47704 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC  
 47705 margin codes to the *posix\_trace\_attr\_setlogfullpolicy()* function.

47706 **Issue 7**

47707 SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the  
 47708 *posix\_trace\_attr\_getstreamfullpolicy()* function declaration.

47709 These functions are marked obsolescent.

## 47710 NAME

47711 posix\_trace\_attr\_getlogsize, posix\_trace\_attr\_getmaxdatasize,  
 47712 posix\_trace\_attr\_getmaxsystemeventsizesize, posix\_trace\_attr\_getmaxusereventsizesize,  
 47713 posix\_trace\_attr\_getstreamsize, posix\_trace\_attr\_setlogsize, posix\_trace\_attr\_setmaxdatasize,  
 47714 posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes (TRACING)

## 47715 SYNOPSIS

```
47716 OB TRC #include <sys/types.h>
47717 #include <trace.h>

47718 TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
47719 size_t *restrict logsize);
47720 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
47721 size_t *restrict maxdatasize);
47722 int posix_trace_attr_getmaxsystemeventsizesize(
47723 const trace_attr_t *restrict attr,
47724 size_t *restrict eventsizesize);
47725 int posix_trace_attr_getmaxusereventsizesize(
47726 const trace_attr_t *restrict attr,
47727 size_t data_len, size_t *restrict eventsizesize);
47728 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
47729 size_t *restrict streamsize);
47730 TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
47731 size_t logsize);
47732 int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
47733 size_t maxdatasize);
47734 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
47735 size_t streamsize);
```

## 47736 DESCRIPTION

47737 TRL The *posix\_trace\_attr\_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*  
 47738 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by  
 47739 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for  
 47740 system and user trace events in the trace log. The default value for the *log-max-size* attribute is  
 47741 implementation-defined.

47742 The *posix\_trace\_attr\_setlogsize()* function shall set the maximum allowed size, in bytes, in the *log-*  
 47743 *max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value  
 47744 supplied by the *logsize* argument.

47745 The trace log size shall be used if the *log-full-policy* attribute is set to `POSIX_TRACE_LOOP` or  
 47746 `POSIX_TRACE_UNTIL_FULL`. If the *log-full-policy* attribute is set to `POSIX_TRACE_APPEND`,  
 47747 the implementation shall ignore the *log-max-size* attribute.

47748 The *posix\_trace\_attr\_getmaxdatasize()* function shall copy the maximum user trace event data  
 47749 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*  
 47750 argument into the variable pointed to by the *maxdatasize* argument. The default value for the  
 47751 *max-data-size* attribute is implementation-defined.

47752 The *posix\_trace\_attr\_getmaxsystemeventsizesize()* function shall calculate the maximum memory size,  
 47753 in bytes, required to store a single system trace event. This value is calculated for the trace  
 47754 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed  
 47755 to by the *eventsizesize* argument.

47756 The values returned as the maximum memory sizes of the user and system trace events shall be  
 47757 such that if the sum of the maximum memory sizes of a set of the trace events that may be

**posix\_trace\_attr\_getlogsize()**

47758 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace  
 47759 stream, the system provides the necessary resources for recording all those trace events, without  
 47760 loss.

47761 The *posix\_trace\_attr\_getmaxusereventsize()* function shall calculate the maximum memory size, in  
 47762 bytes, required to store a single user trace event generated by a call to *posix\_trace\_event()* with a  
 47763 *data\_len* parameter equal to the *data\_len* value specified in this call. This value is calculated for  
 47764 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable  
 47765 pointed to by the *eventsize* argument.

47766 The *posix\_trace\_attr\_getstreamsize()* function shall copy the stream size, in bytes, from the *stream-*  
 47767 *min-size* attribute of the attributes object pointed to by the *attr* argument into the variable  
 47768 pointed to by the *streamsize* argument.

47769 This stream size is the current total memory size reserved for system and user trace events in the  
 47770 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The  
 47771 stream size refers to memory used to store trace event records. Other stream data (for example,  
 47772 trace attribute values) shall not be included in this size.

47773 The *posix\_trace\_attr\_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in  
 47774 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 47775 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size  
 47776 for the user data argument which may be passed to *posix\_trace\_event()*. The implementation  
 47777 shall be allowed to truncate data passed to *trace\_user\_event* which is longer than *maxdatasize*.

47778 The *posix\_trace\_attr\_setstreamsize()* function shall set the minimum allowed size, in bytes, in the  
 47779 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 47780 value supplied by the *streamsize* argument.

**RETURN VALUE**

47781 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 47782 return the corresponding error number.  
 47783

47784 TRL The *posix\_trace\_attr\_getlogsize()* function stores the maximum trace log allowed size in the object  
 47785 pointed to by *logsize*, if successful.

47786 The *posix\_trace\_attr\_getmaxdatasize()* function stores the maximum trace event record memory  
 47787 size in the object pointed to by *maxdatasize*, if successful.

47788 The *posix\_trace\_attr\_getmaxsystemeventsize()* function stores the maximum memory size to store a  
 47789 single system trace event in the object pointed to by *eventsize*, if successful.

47790 The *posix\_trace\_attr\_getmaxusereventsize()* function stores the maximum memory size to store a  
 47791 single user trace event in the object pointed to by *eventsize*, if successful.

47792 The *posix\_trace\_attr\_getstreamsize()* function stores the maximum trace stream allowed size in the  
 47793 object pointed to by *streamsize*, if successful.

**ERRORS**

47794 These functions may fail if:

47795 [EINVAL] The value specified by one of the arguments is invalid.  
 47796

47797 **EXAMPLES**

47798 None.

47799 **APPLICATION USAGE**

47800 None.

47801 **RATIONALE**

47802 None.

47803 **FUTURE DIRECTIONS**

47804 The following functions:

47805 *posix\_trace\_attr\_getlogsize()*  
 47806 *posix\_trace\_attr\_getmaxdatasize()*  
 47807 *posix\_trace\_attr\_getmaxsystemeventszize()*  
 47808 *posix\_trace\_attr\_getmaxusereventszize()*  
 47809 *posix\_trace\_attr\_getstreamsize()*  
 47810 *posix\_trace\_attr\_setlogsize()*  
 47811 *posix\_trace\_attr\_setmaxdatasize()*  
 47812 *posix\_trace\_attr\_setstreamsize()*

47813 may be removed in a future version.

47814 **SEE ALSO**47815 *posix\_trace\_attr\_destroy()*, *posix\_trace\_create()*, *posix\_trace\_event()*, *posix\_trace\_get\_attr()*

47816 XBD &lt;sys/types.h&gt;, &lt;trace.h&gt;

47817 **CHANGE HISTORY**

47818 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47819 **Issue 7**

47820 These functions are marked obsolescent.

**posix\_trace\_attr\_getname()**

System Interfaces

47821 **NAME**47822        posix\_trace\_attr\_getname — retrieve and set information about a trace stream (**TRACING**)47823 **SYNOPSIS**

```
47824 OB TRC #include <trace.h>
47825         int posix_trace_attr_getname(const trace_attr_t *attr,
47826         char *tracename);
```

47827 **DESCRIPTION**47828        Refer to *posix\_trace\_attr\_getclockres()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47829 **NAME**

47830 `posix_trace_attr_getstreamfullpolicy` — retrieve and set the behavior of a trace stream  
47831 **(TRACING)**

47832 **SYNOPSIS**

```
47833 OB TRC #include <trace.h>  
47834 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict  
47835 attr, int *restrict streampolicy);
```

47836 **DESCRIPTION**

47837 Refer to [posix\\_trace\\_attr\\_getinherited\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_getstreamsize()**

System Interfaces

47838 **NAME**47839 `posix_trace_attr_getstreamsize` — retrieve and set trace stream size attributes (TRACING)47840 **SYNOPSIS**

```
47841 OB TRC #include <sys/types.h>  
47842 #include <trace.h>  
  
47843 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,  
47844 size_t *restrict streamsize);
```

47845 **DESCRIPTION**47846 Refer to [posix\\_trace\\_attr\\_getlogsize\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix\_trace\_attr\_init()**47847 **NAME**

47848        posix\_trace\_attr\_init — initialize the trace stream attributes object (TRACING)

47849 **SYNOPSIS**

```
47850 OB TRC #include <trace.h>  
47851        int posix_trace_attr_init(trace_attr_t *attr);
```

47852 **DESCRIPTION**47853        Refer to *posix\_trace\_attr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_setinherited()**

System Interfaces

47854 **NAME**

47855 `posix_trace_attr_setinherited`, `posix_trace_attr_setlogfullpolicy` — retrieve and set the behavior  
47856 of a trace stream (**TRACING**)

47857 **SYNOPSIS**

```
47858 OB TRC #include <trace.h>
47859 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
47860 int inheritancepolicy);
47861 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
47862 int logpolicy);
```

47863 **DESCRIPTION**

47864 Refer to [posix\\_trace\\_attr\\_getinherited\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47865 **NAME**

47866 `posix_trace_attr_setlogsize`, `posix_trace_attr_setmaxdatasize` — retrieve and set trace stream size  
47867 attributes (**TRACING**)

47868 **SYNOPSIS**

```
47869 OB TRC #include <sys/types.h>  
47870 #include <trace.h>  
  
47871 TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,  
47872 size_t logsize);  
47873 OB TRC int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,  
47874 size_t maxdatasize);
```

47875 **DESCRIPTION**

47876 Refer to [posix\\_trace\\_attr\\_getlogsize\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_setname()**

System Interfaces

47877 **NAME**47878            posix\_trace\_attr\_setname — retrieve and set information about a trace stream (**TRACING**)47879 **SYNOPSIS**

```
47880 OB TRC #include <trace.h>
47881         int posix_trace_attr_setname(trace_attr_t *attr,
47882         const char *tracename);
```

47883 **DESCRIPTION**47884            Refer to [posix\\_trace\\_attr\\_getclockres\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47885 **NAME**

47886        **posix\_trace\_attr\_setstreamfullpolicy** — retrieve and set the behavior of a trace stream  
47887        **(TRACING)**

47888 **SYNOPSIS**

```
47889 OB TRC #include <trace.h>  
47890        int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,  
47891        int streampolicy);
```

47892 **DESCRIPTION**

47893        Refer to [posix\\_trace\\_attr\\_getinherited\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_attr\_setstreamsize()**

System Interfaces

47894 **NAME**47895        **posix\_trace\_attr\_setstreamsize** — retrieve and set trace stream size attributes (**TRACING**)47896 **SYNOPSIS**

```
47897 OB TRC #include <sys/types.h>
47898         #include <trace.h>
47899
47899         int posix_trace_attr_setstreamsize(trace_attr_t *attr,
47900         size_t streamsize);
```

47901 **DESCRIPTION**47902        Refer to *posix\_trace\_attr\_getlogsize()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47903 **NAME**47904 `posix_trace_clear` — clear trace stream and trace log (TRACING)47905 **SYNOPSIS**

```
47906 OB TRC #include <sys/types.h>
47907 #include <trace.h>
47908 int posix_trace_clear(trace_id_t trid);
```

47909 **DESCRIPTION**

47910 The `posix_trace_clear()` function shall reinitialize the trace stream identified by the argument `trid`  
 47911 as if it were returning from the `posix_trace_create()` function, except that the same allocated  
 47912 resources shall be reused, the mapping of trace event type identifiers to trace event names shall  
 47913 be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it  
 47914 remains running and if it was suspended, it remains suspended).

47915 All trace events in the trace stream recorded before the call to `posix_trace_clear()` shall be lost. The  
 47916 `posix_stream_full_status` status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee  
 47917 that all trace events that occurred during the `posix_trace_clear()` call are recorded; the behavior  
 47918 with respect to trace points that may occur during this call is unspecified.

47919 OB TRL If the Trace Log option is supported and the trace stream has been created with a log, the  
 47920 `posix_trace_clear()` function shall reinitialize the trace stream with the same behavior as if the  
 47921 trace stream was created without the log, plus it shall reinitialize the trace log associated with  
 47922 the trace stream identified by the argument `trid` as if it were returning from the  
 47923 `posix_trace_create_withlog()` function, except that the same allocated resources, for the trace log,  
 47924 may be reused and the associated trace stream status remains unchanged. The first trace event  
 47925 recorded in the trace log after the call to `posix_trace_clear()` shall be the same as the first trace  
 47926 event recorded in the active trace stream after the call to `posix_trace_clear()`. The  
 47927 `posix_log_full_status` status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that  
 47928 all trace events that occurred during the `posix_trace_clear()` call are recorded in the trace log; the  
 47929 behavior with respect to trace points that may occur during this call is unspecified. If the log full  
 47930 policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace  
 47931 log associated with the trace stream identified by the `trid` argument.

47932 **RETURN VALUE**

47933 Upon successful completion, the `posix_trace_clear()` function shall return a value of zero.  
 47934 Otherwise, it shall return the corresponding error number.

47935 **ERRORS**

47936 The `posix_trace_clear()` function shall fail if:

47937 [EINVAL] The value of the `trid` argument does not correspond to an active trace stream.

47938 **EXAMPLES**

47939 None.

47940 **APPLICATION USAGE**

47941 None.

47942 **RATIONALE**

47943 None.

47944 **FUTURE DIRECTIONS**

47945 The `posix_trace_clear()` function may be removed in a future version.

**posix\_trace\_clear()**

System Interfaces

47946 **SEE ALSO**47947 [posix\\_trace\\_attr\\_destroy\(\)](#), [posix\\_trace\\_create\(\)](#), [posix\\_trace\\_get\\_attr\(\)](#)47948 XBD [<sys/types.h>](#), [<trace.h>](#)47949 **CHANGE HISTORY**

47950 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47951 IEEE PASC Interpretation 1003.1 #123 is applied.

47952 **Issue 7**47953 The *posix\_trace\_clear()* function is marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

47954 **NAME**47955 `posix_trace_close`, `posix_trace_open`, `posix_trace_rewind` — trace log management (TRACING)47956 **SYNOPSIS**

```

47957 OB TRC #include <trace.h>
47958 TRL     int posix_trace_close(trace_id_t trid);
47959         int posix_trace_open(int file_desc, trace_id_t *trid);
47960         int posix_trace_rewind(trace_id_t trid);

```

47961 **DESCRIPTION**

47962 The `posix_trace_close()` function shall deallocate the trace log identifier indicated by *trid*, and all  
 47963 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall  
 47964 fail.

47965 The `posix_trace_open()` function shall allocate the necessary resources and establish the  
 47966 connection between a trace log identified by the *file\_desc* argument and a trace stream identifier  
 47967 identified by the object pointed to by the *trid* argument. The *file\_desc* argument should be a valid  
 47968 open file descriptor that corresponds to a trace log. The *file\_desc* argument shall be open for  
 47969 reading. The current trace event timestamp, which specifies the timestamp of the trace event that  
 47970 will be read by the next call to `posix_trace_getnext_event()`, shall be set to the timestamp of the  
 47971 oldest trace event recorded in the trace log identified by *trid*.

47972 The `posix_trace_open()` function shall return a trace stream identifier in the variable pointed to by  
 47973 the *trid* argument, that may only be used by the following functions:

47974	<code>posix_trace_close()</code>	<code>posix_trace_get_attr()</code>
47975	<code>posix_trace_eventid_equal()</code>	<code>posix_trace_get_status()</code>
47976	<code>posix_trace_eventid_get_name()</code>	<code>posix_trace_getnext_event()</code>
47977	<code>posix_trace_eventtypelist_getnext_id()</code>	<code>posix_trace_rewind()</code>
47978	<code>posix_trace_eventtypelist_rewind()</code>	

47979 In particular, notice that the operations normally used by a trace controller process, such as  
 47980 `posix_trace_start()`, `posix_trace_stop()`, or `posix_trace_shutdown()`, cannot be invoked using the  
 47981 trace stream identifier returned by the `posix_trace_open()` function.

47982 The `posix_trace_rewind()` function shall reset the current trace event timestamp, which specifies  
 47983 the timestamp of the trace event that will be read by the next call to `posix_trace_getnext_event()`,  
 47984 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

47985 **RETURN VALUE**

47986 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 47987 return the corresponding error number.

47988 If successful, the `posix_trace_open()` function stores the trace stream identifier value in the object  
 47989 pointed to by *trid*.

47990 **ERRORS**

47991 The `posix_trace_open()` function shall fail if:

47992	[EINTR]	The operation was interrupted by a signal and thus no trace log was opened.
47993	[EINVAL]	The object pointed to by <i>file_desc</i> does not correspond to a valid trace log.

**posix\_trace\_close()**

System Interfaces

47994 The *posix\_trace\_close()* and *posix\_trace\_rewind()* functions may fail if:  
47995 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

**EXAMPLES**

47996 None.  
47997

**APPLICATION USAGE**

47998 None.  
47999

**RATIONALE**

48000 None.  
48001

**FUTURE DIRECTIONS**

48002 The *posix\_trace\_close()*, *posix\_trace\_open()*, and *posix\_trace\_rewind()* functions may be removed in  
48003 a future version.  
48004

**SEE ALSO**

48005 [posix\\_trace\\_get\\_attr\(\)](#), [posix\\_trace\\_get\\_filter\(\)](#), [posix\\_trace\\_getnext\\_event\(\)](#)  
48006

48007 XBD [<trace.h>](#)

**CHANGE HISTORY**

48008 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.  
48009

48010 IEEE PASC Interpretation 1003.1 #123 is applied.

**Issue 7**

48011 The *posix\_trace\_close()*, *posix\_trace\_open()*, and *posix\_trace\_rewind()* functions are marked  
48012  
48013 obsolescent.

48014 **NAME**

48015 **posix\_trace\_create**, **posix\_trace\_create\_withlog**, **posix\_trace\_flush**, **posix\_trace\_shutdown** —  
 48016 trace stream initialization, flush, and shutdown from a process (TRACING)

48017 **SYNOPSIS**

```
48018 OB TRC #include <sys/types.h>
48019 #include <trace.h>

48020 int posix_trace_create(pid_t pid,
48021     const trace_attr_t *restrict attr,
48022     trace_id_t *restrict trid);
48023 TRL int posix_trace_create_withlog(pid_t pid,
48024     const trace_attr_t *restrict attr, int file_desc,
48025     trace_id_t *restrict trid);
48026 int posix_trace_flush(trace_id_t trid);
48027 int posix_trace_shutdown(trace_id_t trid);
```

48028 **DESCRIPTION**

48029 The *posix\_trace\_create()* function shall create an active trace stream. It allocates all the resources  
 48030 needed by the trace stream being created for tracing the process specified by *pid* in accordance  
 48031 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and  
 48032 shall have been initialized by the function *posix\_trace\_attr\_init()* prior to the *posix\_trace\_create()*  
 48033 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object  
 48034 shall be manipulated through a set of functions described in the *posix\_trace\_attr* family of  
 48035 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the  
 48036 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream  
 48037 shall be set to the value of the system clock, if the Timers option is not supported, or to the value  
 48038 of the CLOCK\_REALTIME clock, if the Timers option is supported.

48039 The *pid* argument represents the target process to be traced. If the process executing this function  
 48040 does not have appropriate privileges to trace the process identified by *pid*, an error shall be  
 48041 returned. If the *pid* argument is zero, the calling process shall be traced.

48042 The *posix\_trace\_create()* function shall store the trace stream identifier of the new trace stream in  
 48043 the object pointed to by the *trid* argument. This trace stream identifier shall be used in  
 48044 subsequent calls to control tracing. The *trid* argument may only be used by the following  
 48045 functions:

48046	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
48047	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
48048	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
48049	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
48050	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
48051	<i>posix_trace_get_attr()</i>	<i>posix_trace_trid_eventid_open()</i>
48052	<i>posix_trace_get_status()</i>	<i>posix_trace_trygetnext_event()</i>

48053 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 48054 argument:

```
48055 posix_trace_get_filter()   posix_trace_set_filter()
```

48056 In particular, notice that the operations normally used by a trace analyzer process, such as  
 48057 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 48058 returned by the *posix\_trace\_create()* function.

**posix\_trace\_create()**

- 48059 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is  
48060 supported, its trace event type filter shall be empty.
- 48061 The *posix\_trace\_create()* function may be called multiple times from the same or different  
48062 processes, with the system-wide limit indicated by the runtime invariant value  
48063 {TRACE\_SYS\_MAX}, which has the minimum value {\_POSIX\_TRACE\_SYS\_MAX}.
- 48064 The trace stream identifier returned by the *posix\_trace\_create()* function in the argument pointed  
48065 to by *trid* is valid only in the process that made the function call. If it is used from another  
48066 process, that is a child process, in functions defined in POSIX.1-2008, these functions shall return  
48067 with the error [EINVAL].
- 48068 TRL The *posix\_trace\_create\_withlog()* function shall be equivalent to *posix\_trace\_create()*, except that it  
48069 associates a trace log with this stream. The *file\_desc* argument shall be the file descriptor  
48070 designating the trace log destination. The function shall fail if this file descriptor refers to a file  
48071 with a file type that is not compatible with the log policy associated with the trace log. The list of  
48072 the appropriate file types that are compatible with each log policy is implementation-defined.
- 48073 The *posix\_trace\_create\_withlog()* function shall return in the parameter pointed to by *trid* the trace  
48074 stream identifier, which uniquely identifies the newly created trace stream, and shall be used in  
48075 subsequent calls to control tracing. The *trid* argument may only be used by the following  
48076 functions:
- |       |   |   |
|-------|---|---|
| 48077 | <i>posix_trace_clear()</i>                    | <i>posix_trace_get_status()</i>         |
| 48078 | <i>posix_trace_eventid_equal()</i>            | <i>posix_trace_getnext_event()</i>      |
| 48079 | <i>posix_trace_eventid_get_name()</i>         | <i>posix_trace_shutdown()</i>           |
| 48080 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_start()</i>              |
| 48081 | <i>posix_trace_eventtypelist_rewind()</i>     | <i>posix_trace_stop()</i>               |
| 48082 | <i>posix_trace_flush()</i>                    | <i>posix_trace_timedgetnext_event()</i> |
| 48083 | <i>posix_trace_get_attr()</i>                 | <i>posix_trace_trid_eventid_open()</i>  |
- 48084 TEF TRL If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
48085 argument:
- 48086 *posix\_trace\_get\_filter()* *posix\_trace\_set\_filter()*
- 48087 TRL In particular, notice that the operations normally used by a trace analyzer process, such as  
48088 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
48089 returned by the *posix\_trace\_create\_withlog()* function.
- 48090 The *posix\_trace\_flush()* function shall initiate a flush operation which copies the contents of the  
48091 trace stream identified by the argument *trid* into the trace log associated with the trace stream at  
48092 the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this  
48093 function shall return an error. The termination of the flush operation can be polled by the  
48094 *posix\_trace\_get\_status()* function. During the flush operation, it shall be possible to trace new  
48095 trace events up to the point when the trace stream becomes full. After flushing is completed, the  
48096 space used by the flushed trace events shall be available for tracing new trace events.
- 48097 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy  
48098 shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:
- 48099 POSIX\_TRACE\_UNTIL\_FULL  
48100 The trace events that have not yet been flushed shall be discarded.

48101		<b>POSIX_TRACE_LOOP</b>
48102		The trace events that have not yet been flushed shall be written to the beginning of the trace log, overwriting previous trace events stored there.
48103		
48104		<b>POSIX_TRACE_APPEND</b>
48105		The trace events that have not yet been flushed shall be appended to the trace log.
48106		The <i>posix_trace_shutdown()</i> function shall stop the tracing of trace events in the trace stream identified by <i>trid</i> , as if <i>posix_trace_stop()</i> had been invoked. The <i>posix_trace_shutdown()</i> function shall free all the resources associated with the trace stream.
48107		
48108		
48109		The <i>posix_trace_shutdown()</i> function shall not return until all the resources associated with the trace stream have been freed. When the <i>posix_trace_shutdown()</i> function returns, the <i>trid</i> argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the <i>trace_getnext_event()</i> functions (which specified this <i>trid</i> ) before this call is unblocked with the error [EINVAL].
48110		
48111		
48112		
48113		
48114		
48115		If the process exits, invokes a member of the <i>exec</i> family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the <i>posix_trace_shutdown()</i> function.
48116		
48117		
48118	TRL	For an active trace stream with log, when the <i>posix_trace_shutdown()</i> function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the <i>posix_trace_flush()</i> function, and the trace log shall be closed.
48119		
48120		
48121		When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.
48122		
48123		
48124		
48125		In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the <i>posix_trace_open()</i> operation.
48126		
48127		The <i>posix_trace_shutdown()</i> function shall not return until all trace events have been flushed.
48128		<b>RETURN VALUE</b>
48129		Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.
48130		
48131	TRL	The <i>posix_trace_create()</i> and <i>posix_trace_create_withlog()</i> functions store the trace stream identifier value in the object pointed to by <i>trid</i> , if successful.
48132		
48133		<b>ERRORS</b>
48134	TRL	The <i>posix_trace_create()</i> and <i>posix_trace_create_withlog()</i> functions shall fail if:
48135		[EAGAIN] No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.
48136		
48137		[EINTR] The operation was interrupted by a signal. No trace stream was created.
48138		[EINVAL] One or more of the trace parameters specified by the <i>attr</i> parameter is invalid.
48139		[ENOMEM] The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.
48140		
48141		[EPERM] The caller does not have appropriate privileges to trace the process specified by <i>pid</i> .
48142		

**posix\_trace\_create()**

48143	[ESRCH]	The <i>pid</i> argument does not refer to an existing process.
48144	TRL	The <i>posix_trace_create_withlog()</i> function shall fail if:
48145	[EBADF]	The <i>file_desc</i> argument is not a valid file descriptor open for writing.
48146	[EINVAL]	The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.
48147		
48148	[ENOSPC]	No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log.
48149		
48150	TRL	The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:
48151	[EINVAL]	The value of the <i>trid</i> argument does not correspond to an active trace stream with log.
48152		
48153	[EFBIG]	The trace log file has attempted to exceed an implementation-defined maximum file size.
48154		
48155	[ENOSPC]	No space left on device.
48156		<b>EXAMPLES</b>
48157		None.
48158		<b>APPLICATION USAGE</b>
48159		None.
48160		<b>RATIONALE</b>
48161		None.
48162		<b>FUTURE DIRECTIONS</b>
48163		The <i>posix_trace_create()</i> , <i>posix_trace_create_withlog()</i> , <i>posix_trace_flush()</i> , and
48164		<i>posix_trace_shutdown()</i> functions may be removed in a future version.
48165		<b>SEE ALSO</b>
48166		<i>clock_getres()</i> , <i>exec</i> , <i>posix_trace_attr_destroy()</i> , <i>posix_trace_clear()</i> , <i>posix_trace_close()</i> ,
48167		<i>posix_trace_eventid_equal()</i> , <i>posix_trace_eventtypelist_getnext_id()</i> , <i>posix_trace_get_attr()</i> ,
48168		<i>posix_trace_get_filter()</i> , <i>posix_trace_getnext_event()</i> , <i>posix_trace_start()</i> , <i>posix_trace_start()</i> , <i>time()</i>
48169		XBD <sys/types.h>, <trace.h>
48170		<b>CHANGE HISTORY</b>
48171		First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
48172		<b>Issue 7</b>
48173		These functions are marked obsolescent.
48174		SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the
48175		<i>posix_trace_trygetnext_event()</i> function from the list of functions that use the <i>trid</i> argument.

48176 **NAME**

48177 posix\_trace\_event, posix\_trace\_eventid\_open — trace functions for instrumenting application  
48178 code (**TRACING**)

48179 **SYNOPSIS**

```
48180 OB TRC #include <sys/types.h>
48181 #include <trace.h>

48182 void posix_trace_event(trace_event_id_t event_id,
48183     const void *restrict data_ptr, size_t data_len);
48184 int posix_trace_eventid_open(const char *restrict event_name,
48185     trace_event_id_t *restrict event_id);
```

48186 **DESCRIPTION**

48187 The *posix\_trace\_event()* function shall record the *event\_id* and the user data pointed to by *data\_ptr*  
48188 in the trace stream into which the calling process is being traced and in which *event\_id* is not  
48189 filtered out. If the total size of the user trace event data represented by *data\_len* is not greater  
48190 than the declared maximum size for user trace event data, then the *truncation-status* attribute of  
48191 the trace event recorded is POSIX\_TRACE\_NOT\_TRUNCATED. Otherwise, the user trace event  
48192 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace  
48193 event recorded is POSIX\_TRACE\_TRUNCATED\_RECORD.

48194 If there is no trace stream created for the process or if the created trace stream is not running, or  
48195 if the trace event specified by *event\_id* is filtered out in the trace stream, the *posix\_trace\_event()*  
48196 function shall have no effect.

48197 The *posix\_trace\_eventid\_open()* function shall associate a user trace event name with a trace event  
48198 type identifier for the calling process. The trace event name is the string pointed to by the  
48199 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
48200 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
48201 trace event type identifiers that can be defined for any given process is limited by the maximum  
48202 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
48203 {POSIX\_TRACE\_USER\_EVENT\_MAX}.

48204 If the Trace Inherit option is not supported, the *posix\_trace\_eventid\_open()* function shall associate  
48205 the user trace event name pointed to by the *event\_name* argument with a trace event type  
48206 identifier that is unique for the traced process, and is returned in the variable pointed to by the  
48207 *event\_id* argument. If the user trace event name has already been mapped for the traced process,  
48208 then the previously assigned trace event type identifier shall be returned. If the per-process user  
48209 trace event name limit represented by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-  
48210 defined POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7, on page 539) user trace event  
48211 shall be returned.

48212 TRI If the Trace Inherit option is supported, the *posix\_trace\_eventid\_open()* function shall associate the  
48213 user trace event name pointed to by the *event\_name* argument with a trace event type identifier  
48214 that is unique for all the processes being traced in this same trace stream, and is returned in the  
48215 variable pointed to by the *event\_id* argument. If the user trace event name has already been  
48216 mapped for the traced processes, then the previously assigned trace event type identifier shall be  
48217 returned. If the per-process user trace event name limit represented by  
48218 {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
48219 POSIX\_TRACE\_UNNAMED\_USEREVENT (Table 2-7, on page 539) user trace event shall be  
48220 returned.

48221 **Note:** The above procedure, together with the fact that multiple processes can only be traced into the  
48222 same trace stream by inheritance, ensure that all the processes that are traced into a trace stream  
48223 have the same mapping of trace event names to trace event type identifiers.

**posix\_trace\_event()**

48224 If there is no trace stream created, the *posix\_trace\_eventid\_open()* function shall store this  
48225 information for future trace streams created for this process.

**RETURN VALUE**

48227 No return value is defined for the *posix\_trace\_event()* function.

48228 Upon successful completion, the *posix\_trace\_eventid\_open()* function shall return a value of zero.  
48229 Otherwise, it shall return the corresponding error number. The *posix\_trace\_eventid\_open()*  
48230 function stores the trace event type identifier value in the object pointed to by *event\_id*, if  
48231 successful.

**ERRORS**

48232 The *posix\_trace\_eventid\_open()* function shall fail if:

48234 [ENAMETOOLONG]

48235 The size of the name pointed to by the *event\_name* argument was longer than  
48236 the implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

**EXAMPLES**

48238 None.

**APPLICATION USAGE**

48240 None.

**RATIONALE**

48242 None.

**FUTURE DIRECTIONS**

48244 The *posix\_trace\_event()* and *posix\_trace\_eventid\_open()* functions may be removed in a future  
48245 version.

**SEE ALSO**

48247 Table 2-7 (on page 539), *exec*, *posix\_trace\_eventid\_equal()*, *posix\_trace\_start()*

48248 XBD <sys/types.h>, <trace.h>

**CHANGE HISTORY**

48250 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48251 IEEE PASC Interpretation 1003.1 #123 is applied.

48252 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of  
48253 the *posix\_trace\_eventid\_open()* function and the *event\_id* argument.

**Issue 7**

48254 The *posix\_trace\_event()* and *posix\_trace\_eventid\_open()* functions are marked obsolescent.  
48255

48256 **NAME**

48257 `posix_trace_eventid_equal`, `posix_trace_eventid_get_name`, `posix_trace_trid_eventid_open` —  
 48258 manipulate the trace event type identifier (**TRACING**)

48259 **SYNOPSIS**

```
48260 OB TRC #include <trace.h>
48261
48261 int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
48262 trace_event_id_t event2);
48263
48263 int posix_trace_eventid_get_name(trace_id_t trid,
48264 trace_event_id_t event, char *event_name);
48265 TEF int posix_trace_trid_eventid_open(trace_id_t trid,
48266 const char *restrict event_name,
48267 trace_event_id_t *restrict event);
```

48268 **DESCRIPTION**

48269 The `posix_trace_eventid_equal()` function shall compare the trace event type identifiers `event1` and  
 48270 `event2` from the same trace stream or the same trace log identified by the `trid` argument. If the  
 48271 trace event type identifiers `event1` and `event2` are from different trace streams, the return value  
 48272 shall be unspecified.

48273 The `posix_trace_eventid_get_name()` function shall return, in the argument pointed to by  
 48274 `event_name`, the trace event name associated with the trace event type identifier identified by the  
 48275 argument `event`, for the trace stream or for the trace log identified by the `trid` argument. The  
 48276 name of the trace event shall have a maximum of `{TRACE_EVENT_NAME_MAX}` characters  
 48277 (which has the minimum value `{_POSIX_TRACE_EVENT_NAME_MAX}`). Successive calls to  
 48278 this function with the same trace event type identifier and the same trace stream identifier shall  
 48279 return the same event name.

48280 TEF The `posix_trace_trid_eventid_open()` function shall associate a user trace event name with a trace  
 48281 event type identifier for a given trace stream. The trace stream is identified by the `trid` argument,  
 48282 and it shall be an active trace stream. The trace event name is the string pointed to by the  
 48283 argument `event_name`. It shall have a maximum of `{TRACE_EVENT_NAME_MAX}` characters  
 48284 (which has the minimum value `{_POSIX_TRACE_EVENT_NAME_MAX}`). The number of user  
 48285 trace event type identifiers that can be defined for any given process is limited by the maximum  
 48286 value `{TRACE_USER_EVENT_MAX}`, which has the minimum value  
 48287 `{_POSIX_TRACE_USER_EVENT_MAX}`.

48288 If the Trace Inherit option is not supported, the `posix_trace_trid_eventid_open()` function shall  
 48289 associate the user trace event name pointed to by the `event_name` argument with a trace event  
 48290 type identifier that is unique for the process being traced in the trace stream identified by the `trid`  
 48291 argument, and is returned in the variable pointed to by the `event` argument. If the user trace  
 48292 event name has already been mapped for the traced process, then the previously assigned trace  
 48293 event type identifier shall be returned. If the per-process user trace event name limit represented  
 48294 by `{TRACE_USER_EVENT_MAX}` has been reached, the pre-defined  
 48295 `POSIX_TRACE_UNNAMED_USEREVENT` (see Table 2-7, on page 539) user trace event shall be  
 48296 returned.

48297 TEF TRI If the Trace Inherit option is supported, the `posix_trace_trid_eventid_open()` function shall  
 48298 associate the user trace event name pointed to by the `event_name` argument with a trace event  
 48299 type identifier that is unique for all the processes being traced in the trace stream identified by  
 48300 the `trid` argument, and is returned in the variable pointed to by the `event` argument. If the user  
 48301 trace event name has already been mapped for the traced processes, then the previously  
 48302 assigned trace event type identifier shall be returned. If the per-process user trace event name  
 48303 limit represented by `{TRACE_USER_EVENT_MAX}` has been reached, the pre-defined

**posix\_trace\_eventid\_equal()**

System Interfaces

48304 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7, on page 539) user trace event shall be  
48305 returned.

**RETURN VALUE**

48307 TEF Upon successful completion, the `posix_trace_eventid_get_name()` and  
48308 `posix_trace_trid_eventid_open()` functions shall return a value of zero. Otherwise, they shall return  
48309 the corresponding error number.

48310 The `posix_trace_eventid_equal()` function shall return a non-zero value if `event1` and `event2` are  
48311 equal; otherwise, a value of zero shall be returned. No errors are defined. If either `event1` or  
48312 `event2` are not valid trace event type identifiers for the trace stream specified by `trid` or if the `trid`  
48313 is invalid, the behavior shall be unspecified.

48314 The `posix_trace_eventid_get_name()` function stores the trace event name value in the object  
48315 pointed to by `event_name`, if successful.

48316 TEF The `posix_trace_trid_eventid_open()` function stores the trace event type identifier value in the  
48317 object pointed to by `event`, if successful.

**ERRORS**

48319 TEF The `posix_trace_eventid_get_name()` and `posix_trace_trid_eventid_open()` functions shall fail if:

48320 [EINVAL] The `trid` argument was not a valid trace stream identifier.

48321 TEF The `posix_trace_trid_eventid_open()` function shall fail if:

48322 TEF [ENAMETOOLONG]

48323 The size of the name pointed to by the `event_name` argument was longer than  
48324 the implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

48325 The `posix_trace_eventid_get_name()` function shall fail if:

48326 [EINVAL] The trace event type identifier `event` was not associated with any name.

**EXAMPLES**

48328 None.

**APPLICATION USAGE**

48330 None.

**RATIONALE**

48332 None.

**FUTURE DIRECTIONS**

48334 The `posix_trace_eventid_equal()`, `posix_trace_eventid_get_name()`, and  
48335 `posix_trace_trid_eventid_open()` functions may be removed in a future version.

**SEE ALSO**

48337 Table 2-7 (on page 539), `exec`, `posix_trace_event()`, `posix_trace_getnext_event()`

48338 XBD <trace.h>

**CHANGE HISTORY**

48340 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48341 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

**Issue 7**

48343 These functions are marked obsolescent.

48344 **NAME**48345        **posix\_trace\_eventid\_open** — trace functions for instrumenting application code (TRACING)48346 **SYNOPSIS**

```
48347 OB TRC #include <sys/types.h>
48348         #include <trace.h>
48349
48349         int posix_trace_eventid_open(const char *restrict event_name,
48350         trace_event_id_t *restrict event_id);
```

48351 **DESCRIPTION**48352        Refer to [posix\\_trace\\_event\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_eventset\_add()**

System Interfaces

48353 **NAME**

48354 `posix_trace_eventset_add`, `posix_trace_eventset_del`, `posix_trace_eventset_empty`,  
 48355 `posix_trace_eventset_fill`, `posix_trace_eventset_ismember` — manipulate trace event type sets  
 48356 (TRACING)

48357 **SYNOPSIS**

```
48358 OB TRC #include <trace.h>
48359 TEF int posix_trace_eventset_add(trace_event_id_t event_id,
48360 trace_event_set_t *set);
48361 int posix_trace_eventset_del(trace_event_id_t event_id,
48362 trace_event_set_t *set);
48363 int posix_trace_eventset_empty(trace_event_set_t *set);
48364 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
48365 int posix_trace_eventset_ismember(trace_event_id_t event_id,
48366 const trace_event_set_t *restrict set, int *restrict ismember);
```

48367 **DESCRIPTION**

48368 These primitives manipulate sets of trace event types. They operate on data objects addressable  
 48369 by the application, not on the current trace event filter of any trace stream.

48370 The `posix_trace_eventset_add()` and `posix_trace_eventset_del()` functions, respectively, shall add or  
 48371 delete the individual trace event type specified by the value of the argument `event_id` to or from  
 48372 the trace event type set pointed to by the argument `set`. Adding a trace event type already in the  
 48373 set or deleting a trace event type not in the set shall not be considered an error.

48374 The `posix_trace_eventset_empty()` function shall initialize the trace event type set pointed to by  
 48375 the `set` argument such that all trace event types defined, both system and user, shall be excluded  
 48376 from the set.

48377 The `posix_trace_eventset_fill()` function shall initialize the trace event type set pointed to by the  
 48378 argument `set`, such that the set of trace event types defined by the argument `what` shall be  
 48379 included in the set. The value of the argument `what` shall consist of one of the following values,  
 48380 as defined in the `<trace.h>` header:

48381 **POSIX\_TRACE\_WOPID\_EVENTS**

48382 All the process-independent implementation-defined system trace event types are included  
 48383 in the set.

48384 **POSIX\_TRACE\_SYSTEM\_EVENTS**

48385 All the implementation-defined system trace event types are included in the set, as are those  
 48386 defined in POSIX.1-2008.

48387 **POSIX\_TRACE\_ALL\_EVENTS**

48388 All trace event types defined, both system and user, are included in the set.

48389 Applications shall call either `posix_trace_eventset_empty()` or `posix_trace_eventset_fill()` at least  
 48390 once for each object of type `trace_event_set_t` prior to any other use of that object. If such an  
 48391 object is not initialized in this way, but is nonetheless supplied as an argument to any of the  
 48392 `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, or `posix_trace_eventset_ismember()` functions,  
 48393 the results are undefined.

48394 The `posix_trace_eventset_ismember()` function shall test whether the trace event type specified by  
 48395 the value of the argument `event_id` is a member of the set pointed to by the argument `set`. The  
 48396 value returned in the object pointed to by `ismember` argument is zero if the trace event type  
 48397 identifier is not a member of the set and a value different from zero if it is a member of the set.

**48398 RETURN VALUE**

48399       Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
48400       return the corresponding error number.

**48401 ERRORS**

48402       These functions may fail if:

48403       [EINVAL]       The value of one of the arguments is invalid.

**48404 EXAMPLES**

48405       None.

**48406 APPLICATION USAGE**

48407       None.

**48408 RATIONALE**

48409       None.

**48410 FUTURE DIRECTIONS**

48411       The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,  
48412       `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions may be removed in a  
48413       future version.

**48414 SEE ALSO**

48415       [\*posix\\_trace\\_eventid\\_equal\(\)\*](#), [\*posix\\_trace\\_get\\_filter\(\)\*](#)

48416       XBD <[trace.h](#)>

**48417 CHANGE HISTORY**

48418       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

**48419 Issue 7**

48420       The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,  
48421       `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions are marked obsolescent.

**posix\_trace\_eventtypelist\_getnext\_id()**

System Interfaces

48422 **NAME**

48423 `posix_trace_eventtypelist_getnext_id`, `posix_trace_eventtypelist_rewind` — iterate over a  
 48424 mapping of trace event types (**TRACING**)

48425 **SYNOPSIS**

```
48426 OB TRC #include <trace.h>
48427
48427 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
48428     trace_event_id_t *restrict event, int *restrict unavailable);
48429 int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

48430 **DESCRIPTION**

48431 The first time `posix_trace_eventtypelist_getnext_id()` is called, the function shall return in the  
 48432 variable pointed to by `event` the first trace event type identifier of the list of trace events of the  
 48433 trace stream identified by the `trid` argument. Successive calls to  
 48434 `posix_trace_eventtypelist_getnext_id()` return in the variable pointed to by `event` the next trace  
 48435 event type identifier in that same list. Each time a trace event type identifier is successfully  
 48436 written into the variable pointed to by the `event` argument, the variable pointed to by the  
 48437 `unavailable` argument shall be set to zero. When no more trace event type identifiers are available,  
 48438 and so none is returned, the variable pointed to by the `unavailable` argument shall be set to a  
 48439 value different from zero.

48440 The `posix_trace_eventtypelist_rewind()` function shall reset the next trace event type identifier to  
 48441 be read to the first trace event type identifier from the list of trace events used in the trace stream  
 48442 identified by `trid`.

48443 **RETURN VALUE**

48444 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 48445 return the corresponding error number.

48446 The `posix_trace_eventtypelist_getnext_id()` function stores the trace event type identifier value in  
 48447 the object pointed to by `event`, if successful.

48448 **ERRORS**

48449 These functions shall fail if:

48450 [EINVAL] The `trid` argument was not a valid trace stream identifier.

48451 **EXAMPLES**

48452 None.

48453 **APPLICATION USAGE**

48454 None.

48455 **RATIONALE**

48456 None.

48457 **FUTURE DIRECTIONS**

48458 The `posix_trace_eventtypelist_getnext_id()` and `posix_trace_eventtypelist_rewind()` functions may be  
 48459 removed in a future version.

48460 **SEE ALSO**

48461 [posix\\_trace\\_event\(\)](#), [posix\\_trace\\_eventid\\_equal\(\)](#), [posix\\_trace\\_getnext\\_event\(\)](#)

48462 XBD [<trace.h>](#)

48463 **CHANGE HISTORY**

48464 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48465 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

48466 **Issue 7**48467 The *posix\_trace\_eventtypelist\_getnext\_id()* and *posix\_trace\_eventtypelist\_rewind()* functions are  
48468 marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_flush()**

System Interfaces

48469 **NAME**48470 `posix_trace_flush` — trace stream flush from a process (**TRACING**)48471 **SYNOPSIS**

```
48472 OB TRC #include <sys/types.h>  
48473 #include <trace.h>  
48474 TRL int posix_trace_flush(trace_id_t trid);
```

48475 **DESCRIPTION**48476 Refer to [posix\\_trace\\_create\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

48477 **NAME**

48478 `posix_trace_get_attr`, `posix_trace_get_status` — retrieve the trace attributes or trace status  
 48479 (TRACING)

48480 **SYNOPSIS**

```
48481 OB TRC #include <trace.h>
48482
48482 int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
48483 int posix_trace_get_status(trace_id_t trid,
48484 struct posix_trace_status_info *statusinfo);
```

48485 **DESCRIPTION**

48486 The `posix_trace_get_attr()` function shall copy the attributes of the active trace stream identified  
 48487 TRL by `trid` into the object pointed to by the `attr` argument. If the Trace Log option is supported, `trid`  
 48488 may represent a pre-recorded trace log.

48489 The `posix_trace_get_status()` function shall return, in the structure pointed to by the `statusinfo`  
 48490 argument, the current trace status for the trace stream identified by the `trid` argument. These  
 48491 status values returned in the structure pointed to by `statusinfo` shall have been appropriately  
 48492 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and  
 48493 the `trid` argument refers to a pre-recorded trace stream, the status shall be the status of the  
 48494 completed trace stream.

48495 Each time the `posix_trace_get_status()` function is used, the overrun status of the trace stream  
 48496 TRL shall be reset to `POSIX_TRACE_NO_OVERRUN` immediately after the call completes. If the  
 48497 Trace Log option is supported, the `posix_trace_get_status()` function shall behave the same as  
 48498 when the option is not supported except for the following differences:

- 48499 • If the `trid` argument refers to a trace stream with log, each time the `posix_trace_get_status()`  
 48500 function is used, the log overrun status of the trace stream shall be reset to  
 48501 `POSIX_TRACE_NO_OVERRUN` and the `flush_error` status shall be reset to zero  
 48502 immediately after the call completes.
- 48503 • If the `trid` argument refers to a pre-recorded trace stream, the status returned shall be the  
 48504 status of the completed trace stream and the status values of the trace stream shall not be  
 48505 reset.

48506 **RETURN VALUE**

48507 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 48508 return the corresponding error number.

48509 The `posix_trace_get_attr()` function stores the trace attributes in the object pointed to by `attr`, if  
 48510 successful.

48511 The `posix_trace_get_status()` function stores the trace status in the object pointed to by `statusinfo`,  
 48512 if successful.

48513 **ERRORS**

48514 These functions shall fail if:

- 48515 [EINVAL] The trace stream argument `trid` does not correspond to a valid active trace  
 48516 stream or a valid trace log.

**posix\_trace\_get\_attr()**48517 **EXAMPLES**

48518 None.

48519 **APPLICATION USAGE**

48520 None.

48521 **RATIONALE**

48522 None.

48523 **FUTURE DIRECTIONS**48524 The *posix\_trace\_get\_attr()* and *posix\_trace\_get\_status()* functions may be removed in a future  
48525 version.48526 **SEE ALSO**48527 [posix\\_trace\\_attr\\_destroy\(\)](#), [posix\\_trace\\_close\(\)](#), [posix\\_trace\\_create\(\)](#)48528 XBD [<trace.h>](#)48529 **CHANGE HISTORY**

48530 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48531 IEEE PASC Interpretation 1003.1 #123 is applied.

48532 **Issue 7**48533 The *posix\_trace\_get\_attr()* and *posix\_trace\_get\_status()* functions are marked obsolescent.

48534 **NAME**

48535 `posix_trace_get_filter`, `posix_trace_set_filter` — retrieve and set the filter of an initialized trace  
 48536 stream (**TRACING**)

48537 **SYNOPSIS**

```
48538 OB TRC #include <trace.h>
48539 TEF int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
48540 int posix_trace_set_filter(trace_id_t trid,
48541 const trace_event_set_t *set, int how);
```

48542 **DESCRIPTION**

48543 The `posix_trace_get_filter()` function shall retrieve, into the argument pointed to by `set`, the actual  
 48544 trace event filter from the trace stream specified by `trid`.

48545 The `posix_trace_set_filter()` function shall change the set of filtered trace event types after a trace  
 48546 stream identified by the `trid` argument is created. This function may be called prior to starting  
 48547 the trace stream, or while the trace stream is active. By default, if no call is made to  
 48548 `posix_trace_set_filter()`, all trace events shall be recorded (that is, none of the trace event types are  
 48549 filtered out).

48550 If this function is called while the trace is in progress, a special system trace event,  
 48551 `POSIX_TRACE_FILTER`, shall be recorded in the trace indicating both the old and the new sets  
 48552 of filtered trace event types (see [Table 2-4](#) (on page 537) and [Table 2-6](#), on page 538).

48553 If the `posix_trace_set_filter()` function is interrupted by a signal, an error shall be returned and the  
 48554 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

48555 The value of the argument `how` indicates the manner in which the set is to be changed and shall  
 48556 have one of the following values, as defined in the `<trace.h>` header:

48557 `POSIX_TRACE_SET_EVENTSET`

48558 The resulting set of trace event types to be filtered shall be the trace event type set pointed  
 48559 to by the argument `set`.

48560 `POSIX_TRACE_ADD_EVENTSET`

48561 The resulting set of trace event types to be filtered shall be the union of the current set and  
 48562 the trace event type set pointed to by the argument `set`.

48563 `POSIX_TRACE_SUB_EVENTSET`

48564 The resulting set of trace event types to be filtered shall be all trace event types in the  
 48565 current set that are not in the set pointed to by the argument `set`; that is, remove each  
 48566 element of the specified set from the current filter.

48567 **RETURN VALUE**

48568 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 48569 return the corresponding error number.

48570 The `posix_trace_get_filter()` function stores the set of filtered trace event types in `set`, if successful.

48571 **ERRORS**

48572 These functions shall fail if:

48573 `[EINVAL]` The value of the `trid` argument does not correspond to an active trace stream  
 48574 or the value of the argument pointed to by `set` is invalid.

48575 `[EINTR]` The operation was interrupted by a signal.

**posix\_trace\_get\_filter()**

System Interfaces

48576 **EXAMPLES**

48577       None.

48578 **APPLICATION USAGE**

48579       None.

48580 **RATIONALE**

48581       None.

48582 **FUTURE DIRECTIONS**48583       The *posix\_trace\_get\_filter()* and *posix\_trace\_set\_filter()* functions may be removed in a future  
48584       version.48585 **SEE ALSO**48586       [Table 2-4](#) (on page 537), [Table 2-6](#) (on page 538), [posix\\_trace\\_eventset\\_add\(\)](#)48587       XBD [<trace.h>](#)48588 **CHANGE HISTORY**

48589       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48590       IEEE PASC Interpretation 1003.1 #123 is applied.

48591 **Issue 7**48592       The *posix\_trace\_get\_filter()* and *posix\_trace\_set\_filter()* functions are marked obsolescent.

48593 **NAME**48594        **posix\_trace\_get\_status** — retrieve the trace status (**TRACING**)48595 **SYNOPSIS**

```
48596 OB TRC #include <trace.h>
48597         int posix_trace_get_status(trace_id_t trid,
48598         struct posix_trace_status_info *statusinfo);
```

48599 **DESCRIPTION**48600        Refer to *posix\_trace\_get\_attr()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_getnext\_event()**

System Interfaces

48601 **NAME**

48602 `posix_trace_getnext_event`, `posix_trace_timedgetnext_event`, `posix_trace_trygetnext_event` —  
 48603 retrieve a trace event (**TRACING**)

48604 **SYNOPSIS**

```
48605 OB TRC #include <sys/types.h>
48606 #include <trace.h>

48607 int posix_trace_getnext_event(trace_id_t trid,
48608     struct posix_trace_event_info *restrict event,
48609     void *restrict data, size_t num_bytes,
48610     size_t *restrict data_len, int *restrict unavailable);
48611 int posix_trace_timedgetnext_event(trace_id_t trid,
48612     struct posix_trace_event_info *restrict event,
48613     void *restrict data, size_t num_bytes,
48614     size_t *restrict data_len, int *restrict unavailable,
48615     const struct timespec *restrict abstime);
48616 int posix_trace_trygetnext_event(trace_id_t trid,
48617     struct posix_trace_event_info *restrict event,
48618     void *restrict data, size_t num_bytes,
48619     size_t *restrict data_len, int *restrict unavailable);
```

48620 **DESCRIPTION**

48621 The `posix_trace_getnext_event()` function shall report a recorded trace event either from an active  
 48622 TRL trace stream without log or a pre-recorded trace stream identified by the `trid` argument. The  
 48623 `posix_trace_trygetnext_event()` function shall report a recorded trace event from an active trace  
 48624 stream without log identified by the `trid` argument.

48625 The trace event information associated with the recorded trace event shall be copied by the  
 48626 function into the structure pointed to by the argument `event` and the data associated with the  
 48627 trace event shall be copied into the buffer pointed to by the `data` argument.

48628 The `posix_trace_getnext_event()` function shall block if the `trid` argument identifies an active trace  
 48629 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded  
 48630 trace event was reported, the variable pointed to by the `unavailable` argument shall be set to zero.  
 48631 Otherwise, the variable pointed to by the `unavailable` argument shall be set to a value different  
 48632 from zero.

48633 The `posix_trace_timedgetnext_event()` function shall attempt to get another trace event from an  
 48634 active trace stream without log, as in the `posix_trace_getnext_event()` function. However, if no  
 48635 trace event is available from the trace stream, the implied wait shall be terminated when the  
 48636 timeout specified by the argument `abstime` expires, and the function shall return the error  
 48637 [ETIMEDOUT].

48638 The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the  
 48639 clock upon which timeouts are based (that is, when the value of that clock equals or exceeds  
 48640 `abstime`), or if the absolute time specified by `abstime` has already passed at the time of the call.

48641 The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall  
 48642 be the resolution of the clock on which it is based. The `timespec` data type is defined in the  
 48643 `<time.h>` header.

48644 Under no circumstance shall the function fail with a timeout if a trace event is immediately  
 48645 available from the trace stream. The validity of the `abstime` argument need not be checked if a  
 48646 trace event is immediately available from the trace stream.

- 48647 The behavior of this function for a pre-recorded trace stream is unspecified.
- 48648 TRL The *posix\_trace\_trygetnext\_event()* function shall not block. This function shall return an error if  
 48649 the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported,  
 48650 the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace  
 48651 event was reported, the variable pointed to by the *unavailable* argument shall be set to a value  
 48652 different from zero.
- 48653 The argument *num\_bytes* shall be the size of the buffer pointed to by the *data* argument. The  
 48654 argument *data\_len* reports to the application the length in bytes of the data record just  
 48655 transferred. If *num\_bytes* is greater than or equal to the size of the data associated with the trace  
 48656 event pointed to by the *event* argument, all the recorded data shall be transferred. In this case,  
 48657 the *truncation-status* member of the trace event structure shall be either  
 48658 POSIX\_TRACE\_NOT\_TRUNCATED, if the trace event data was recorded without truncation  
 48659 while tracing, or POSIX\_TRACE\_TRUNCATED\_RECORD, if the trace event data was truncated  
 48660 when it was recorded. If the *num\_bytes* argument is less than the length of recorded trace event  
 48661 data, the data transferred shall be truncated to a length of *num\_bytes*, the value stored in the  
 48662 variable pointed to by *data\_len* shall be equal to *num\_bytes*, and the *truncation-status* member of  
 48663 the *event* structure argument shall be set to POSIX\_TRACE\_TRUNCATED\_READ (see the  
 48664 **posix\_trace\_event\_info** structure defined in <trace.h>).
- 48665 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace  
 48666 events shall be reported in the order in which they were generated, up to an implementation-  
 48667 defined time resolution that causes the ordering of trace events occurring very close to each  
 48668 other to be unknown. Once reported, a trace event cannot be reported again from an active trace  
 48669 stream. Once a trace event is reported from an active trace stream without log, the trace stream  
 48670 shall make the resources associated with that trace event available to record future generated  
 48671 trace events.
- 48672 **RETURN VALUE**
- 48673 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 48674 return the corresponding error number.
- 48675 If successful, these functions store:
- 48676 • The recorded trace event in the object pointed to by *event*
  - 48677 • The trace event information associated with the recorded trace event in the object pointed  
 48678 to by *data*
  - 48679 • The length of this trace event information in the object pointed to by *data\_len*
  - 48680 • The value of zero in the object pointed to by *unavailable*
- 48681 **ERRORS**
- 48682 These functions shall fail if:
- 48683 [EINVAL] The trace stream identifier argument *trid* is invalid.
- 48684 The *posix\_trace\_getnext\_event()* and *posix\_trace\_timedgetnext\_event()* functions shall fail if:
- 48685 [EINTR] The operation was interrupted by a signal, and so the call had no effect.
- 48686 The *posix\_trace\_trygetnext\_event()* function shall fail if:
- 48687 [EINVAL] The trace stream identifier argument *trid* does not correspond to an active  
 48688 trace stream.

**posix\_trace\_getnext\_event()**

System Interfaces

48689 The *posix\_trace\_timedgetnext\_event()* function shall fail if:

48690 [EINVAL] There is no trace event immediately available from the trace stream, and the  
48691 *timeout* argument is invalid.

48692 [ETIMEDOUT] No trace event was available from the trace stream before the specified  
48693 *timeout* expired.

48694 **EXAMPLES**

48695 None.

48696 **APPLICATION USAGE**

48697 None.

48698 **RATIONALE**

48699 None.

48700 **FUTURE DIRECTIONS**

48701 These functions may be removed in a future version.

48702 **SEE ALSO**

48703 *posix\_trace\_close()*, *posix\_trace\_create()*

48704 XBD <*sys/types.h*>, <*trace.h*>

48705 **CHANGE HISTORY**

48706 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48707 IEEE PASC Interpretation 1003.1 #123 is applied.

48708 **Issue 7**

48709 The *posix\_trace\_getnext\_event()*, *posix\_trace\_timedgetnext\_event()*, and  
48710 *posix\_trace\_trygetnext\_event()* functions are marked obsolescent.

48711 Functionality relating to the Timers option is moved to the Base.

*System Interfaces***posix\_trace\_open()**48712 **NAME**

48713        posix\_trace\_open, posix\_trace\_rewind — trace log management (TRACING)

48714 **SYNOPSIS**

48715 OB TRC   #include &lt;trace.h&gt;

48716 TRL       int posix\_trace\_open(int *file\_desc*, trace\_id\_t \**trid*);48717       int posix\_trace\_rewind(trace\_id\_t *trid*);48718 **DESCRIPTION**48719       Refer to [posix\\_trace\\_close\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_set\_filter()**

System Interfaces

48720 **NAME**48721            posix\_trace\_set\_filter — set filter of an initialized trace stream (**TRACING**)48722 **SYNOPSIS**

48723 OB TRC    #include &lt;trace.h&gt;

48724 TEF        int posix\_trace\_set\_filter(trace\_id\_t trid,  
48725            const trace\_event\_set\_t \*set, int how);48726 **DESCRIPTION**48727            Refer to [posix\\_trace\\_get\\_filter\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

48728 **NAME**48729 `posix_trace_shutdown` — trace stream shutdown from a process (TRACING)48730 **SYNOPSIS**

```
48731 OB TRC #include <sys/types.h>  
48732 #include <trace.h>  
48733 int posix_trace_shutdown(trace_id_t trid);
```

48734 **DESCRIPTION**48735 Refer to [posix\\_trace\\_create\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_start()**48736 **NAME**48737            **posix\_trace\_start, posix\_trace\_stop** — trace start and stop (**TRACING**)48738 **SYNOPSIS**

```
48739 OB TRC #include <trace.h>
48740
48740 int posix_trace_start(trace_id_t trid);
48741 int posix_trace_stop (trace_id_t trid);
```

48742 **DESCRIPTION**

48743        The *posix\_trace\_start()* and *posix\_trace\_stop()* functions, respectively, shall start and stop the trace  
48744 stream identified by the argument *trid*.

48745        The effect of calling the *posix\_trace\_start()* function shall be recorded in the trace stream as the  
48746 POSIX\_TRACE\_START system trace event and the status of the trace stream shall become  
48747 POSIX\_TRACE\_RUNNING. If the trace stream is in progress when this function is called, the  
48748 POSIX\_TRACE\_START system trace event shall not be recorded and the trace stream shall  
48749 continue to run. If the trace stream is full, the POSIX\_TRACE\_START system trace event shall  
48750 not be recorded and the status of the trace stream shall not be changed.

48751        The effect of calling the *posix\_trace\_stop()* function shall be recorded in the trace stream as the  
48752 POSIX\_TRACE\_STOP system trace event and the status of the trace stream shall become  
48753 POSIX\_TRACE\_SUSPENDED. If the trace stream is suspended when this function is called, the  
48754 POSIX\_TRACE\_STOP system trace event shall not be recorded and the trace stream shall remain  
48755 suspended. If the trace stream is full, the POSIX\_TRACE\_STOP system trace event shall not be  
48756 recorded and the status of the trace stream shall not be changed.

48757 **RETURN VALUE**

48758        Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
48759 return the corresponding error number.

48760 **ERRORS**

48761        These functions shall fail if:

48762        [EINVAL]        The value of the argument *trid* does not correspond to an active trace stream  
48763 and thus no trace stream was started or stopped.

48764        [EINTR]        The operation was interrupted by a signal and thus the trace stream was not  
48765 necessarily started or stopped.

48766 **EXAMPLES**

48767        None.

48768 **APPLICATION USAGE**

48769        None.

48770 **RATIONALE**

48771        None.

48772 **FUTURE DIRECTIONS**

48773        The *posix\_trace\_start()* and *posix\_trace\_stop()* functions may be removed in a future version.

48774 **SEE ALSO**

48775        *posix\_trace\_create()*

48776        XBD [<trace.h>](#)

48777 **CHANGE HISTORY**

48778 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48779 IEEE PASC Interpretation 1003.1 #123 is applied.

48780 **Issue 7**48781 The *posix\_trace\_start()* and *posix\_trace\_stop()* functions are marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_timedgetnext\_event()**

System Interfaces

48782 **NAME**48783        **posix\_trace\_timedgetnext\_event** — retrieve a trace event (TRACING)48784 **SYNOPSIS**

```
48785 OB TRC #include <sys/types.h>
48786         #include <trace.h>

48787         int posix_trace_timedgetnext_event(trace_id_t trid,
48788         struct posix_trace_event_info *restrict event,
48789         void *restrict data, size_t num_bytes,
48790         size_t *restrict data_len, int *restrict unavailable,
48791         const struct timespec *restrict abstime);
```

48792 **DESCRIPTION**48793        Refer to [posix\\_trace\\_getnext\\_event\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

48794 **NAME**

48795        posix\_trace\_trid\_eventid\_open — open a trace event type identifier (TRACING)

48796 **SYNOPSIS**

```
48797 OB TRC #include <trace.h>
48798 TEF     int posix_trace_trid_eventid_open(trace_id_t trid,
48799         const char *restrict event_name,
48800         trace_event_id_t *restrict event);
```

48801 **DESCRIPTION**48802        Refer to [posix\\_trace\\_eventid\\_equal\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**posix\_trace\_trygetnext\_event()**

System Interfaces

48803 **NAME**48804 `posix_trace_trygetnext_event` — retrieve a trace event (**TRACING**)48805 **SYNOPSIS**

```
48806 OB TRC #include <sys/types.h>
48807 #include <trace.h>

48808 int posix_trace_trygetnext_event(trace_id_t trid,
48809 struct posix_trace_event_info *restrict event,
48810 void *restrict data, size_t num_bytes,
48811 size_t *restrict data_len, int *restrict unavailable);
```

48812 **DESCRIPTION**48813 Refer to [posix\\_trace\\_getnext\\_event\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

48814 **NAME**48815 `posix_typed_mem_get_info` — query typed memory information (**ADVANCED REALTIME**)48816 **SYNOPSIS**

```
48817 TYM      #include <sys/mman.h>
48818
48818      int posix_typed_mem_get_info(int fildes,
48819      struct posix_typed_mem_info *info);
```

48820 **DESCRIPTION**

48821 The `posix_typed_mem_get_info()` function shall return, in the `posix_tmi_length` field of the  
 48822 **posix\_typed\_mem\_info** structure pointed to by `info`, the maximum length which may be  
 48823 successfully allocated by the typed memory object designated by `fildes`. This maximum length  
 48824 shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or  
 48825 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object  
 48826 represented by `fildes` was opened. The maximum length is dynamic; therefore, the value  
 48827 returned is valid only while the current mapping of the corresponding typed memory pool  
 48828 remains unchanged.

48829 If `fildes` represents a typed memory object opened with neither the  
 48830 `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`  
 48831 flag specified, the returned value of `info->posix_tmi_length` is unspecified.

48832 The `posix_typed_mem_get_info()` function may return additional implementation-defined  
 48833 information in other fields of the **posix\_typed\_mem\_info** structure pointed to by `info`.

48834 If the memory object specified by `fildes` is not a typed memory object, then the behavior of this  
 48835 function is undefined.

48836 **RETURN VALUE**

48837 Upon successful completion, the `posix_typed_mem_get_info()` function shall return zero;  
 48838 otherwise, the corresponding error status value shall be returned.

48839 **ERRORS**

48840 The `posix_typed_mem_get_info()` function shall fail if:

48841 [EBADF] The `fildes` argument is not a valid open file descriptor.

48842 [ENODEV] The `fildes` argument is not connected to a memory object supported by this  
 48843 function.

48844 This function shall not return an error code of [EINTR].

48845 **EXAMPLES**

48846 None.

48847 **APPLICATION USAGE**

48848 None.

48849 **RATIONALE**

48850 An application that needs to allocate a block of typed memory with length dependent upon the  
 48851 amount of memory currently available must either query the typed memory object to obtain the  
 48852 amount available, or repeatedly invoke `mmap()` attempting to guess an appropriate length.  
 48853 While the latter method is existing practice with `malloc()`, it is awkward and imprecise. The  
 48854 `posix_typed_mem_get_info()` function allows an application to immediately determine available  
 48855 memory. This is particularly important for typed memory objects that may in some cases be  
 48856 scarce resources. Note that when a typed memory pool is a shared resource, some form of  
 48857 mutual-exclusion or synchronization may be required while typed memory is being queried and

**posix\_typed\_mem\_get\_info()**

System Interfaces

48858 allocated to prevent race conditions.

48859 The existing *fstat()* function is not suitable for this purpose. We realize that implementations  
48860 may wish to provide other attributes of typed memory objects (for example, alignment  
48861 requirements, page size, and so on). The *fstat()* function returns a structure which is not  
48862 extensible and, furthermore, contains substantial information that is inappropriate for typed  
48863 memory objects.

**48864 FUTURE DIRECTIONS**

48865 None.

**48866 SEE ALSO**

48867 *fstat()*, *mmap()*, *posix\_typed\_mem\_open()*

48868 XBD <sys/mman.h>

**48869 CHANGE HISTORY**

48870 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

48871 **NAME**

48872 posix\_typed\_mem\_open — open a typed memory object (ADVANCED REALTIME)

48873 **SYNOPSIS**

```
48874 TYM #include <sys/mman.h>
48875 int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

48876 **DESCRIPTION**

48877 The *posix\_typed\_mem\_open()* function shall establish a connection between the typed memory  
 48878 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file  
 48879 description that refers to the typed memory object and a file descriptor that refers to that open  
 48880 file description. The file descriptor is used by other functions to refer to that typed memory  
 48881 object. It is unspecified whether the name appears in the file system and is visible to other  
 48882 functions that take pathnames as arguments. The *name* argument conforms to the construction  
 48883 rules for a pathname, except that the interpretation of <slash> characters other than the leading  
 48884 <slash> character in *name* is implementation-defined, and that the length limits for the *name*  
 48885 argument are implementation-defined and need not be the same as the pathname limits  
 48886 {PATH\_MAX} and {NAME\_MAX}. If *name* begins with the <slash> character, then processes  
 48887 calling *posix\_typed\_mem\_open()* with the same value of *name* shall refer to the same typed  
 48888 memory object. If *name* does not begin with the <slash> character, the effect is implementation-  
 48889 defined.

48890 Each typed memory object supported in a system shall be identified by a name which specifies  
 48891 not only its associated typed memory pool, but also the path or port by which it is accessed. That  
 48892 is, the same typed memory pool accessed via several different ports shall have several different  
 48893 corresponding names. The binding between names and typed memory objects is established in  
 48894 an implementation-defined manner. Unlike shared memory objects, there is no way within  
 48895 POSIX.1-2008 for a program to create a typed memory object.

48896 The value of *tflag* shall determine how the typed memory object behaves when subsequently  
 48897 mapped by calls to *mmap()*. At most, one of the following flags defined in <sys/mman.h> may  
 48898 be specified:

48899 POSIX\_TYPED\_MEM\_ALLOCATE

48900 Allocate on *mmap()*.

48901 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG

48902 Allocate contiguously on *mmap()*.

48903 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE

48904 Map on *mmap()*, without affecting allocatability.

48905 If *tflag* has the flag POSIX\_TYPED\_MEM\_ALLOCATE specified, any subsequent call to *mmap()*  
 48906 using the returned file descriptor shall result in allocation and mapping of typed memory from  
 48907 the specified typed memory pool. The allocated memory may be a contiguous previously  
 48908 unallocated area of the typed memory pool or several non-contiguous previously unallocated  
 48909 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag  
 48910 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 48911 returned file descriptor shall result in allocation and mapping of a single contiguous previously  
 48912 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process  
 48913 address space). If *tflag* has none of the flags POSIX\_TYPED\_MEM\_ALLOCATE or  
 48914 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 48915 returned file descriptor shall map an application-chosen area from the specified typed memory  
 48916 pool such that this mapped area becomes unavailable for allocation until unmapped by all  
 48917 processes. If *tflag* has the flag POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE specified, any

**posix\_typed\_mem\_open()**

48918 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen  
 48919 area from the specified typed memory pool without an effect on the availability of that area for  
 48920 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it  
 48921 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping.  
 48922 Appropriate privileges to specify the POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag are  
 48923 implementation-defined.

48924 If successful, *posix\_typed\_mem\_open()* shall return a file descriptor for the typed memory object  
 48925 that is the lowest numbered file descriptor not currently open for that process. The open file  
 48926 description is new, and therefore the file descriptor shall not share it with any other processes. It  
 48927 is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated  
 48928 with the new file descriptor shall be cleared.

48929 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,  
 48930 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified  
 48931 when passed a file descriptor connected to a typed memory object by this function.

48932 The file status flags of the open file description shall be set according to the value of *oflag*.  
 48933 Applications shall specify exactly one of the three access mode values described below and  
 48934 defined in the `<fcntl.h>` header, as the value of *oflag*.

48935 O\_RDONLY Open for read access only.

48936 O\_WRONLY Open for write access only.

48937 O\_RDWR Open for read or write access.

**RETURN VALUE**

48938 Upon successful completion, the *posix\_typed\_mem\_open()* function shall return a non-negative  
 48939 integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1`  
 48940 and set *errno* to indicate the error.  
 48941

**ERRORS**

48942 The *posix\_typed\_mem\_open()* function shall fail if:

48943 [EACCES] The typed memory object exists and the permissions specified by *oflag* are  
 48944 denied.

48945 [EINTR] The *posix\_typed\_mem\_open()* operation was interrupted by a signal.

48946 [EINVAL] The flags specified in *tflag* are invalid (more than one of  
 48947 POSIX\_TYPED\_MEM\_ALLOCATE,  
 48948 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG, or  
 48949 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE is specified).  
 48950

48951 [EMFILE] All file descriptors available to the process are currently open.

48952 [ENAMETOOLONG]

48953 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems  
 48954 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI  
 48955 systems, or has a pathname component that is longer than  
 48956 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or  
 48957 longer than `{_XOPEN_NAME_MAX}` on XSI systems.

48958 [ENFILE] Too many file descriptors are currently open in the system.

48959 [ENOENT] The named typed memory object does not exist.

48960 [E`PERM`] The caller lacks appropriate privileges to specify the  
 48961 `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flag in the *tflag* argument.

**EXAMPLES**

48962 None.

**APPLICATION USAGE**

48965 None.

**RATIONALE**

48967 None.

**FUTURE DIRECTIONS**

48969 None.

**SEE ALSO**

48971 *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*, *posix\_mem\_offset()*,  
 48972 *posix\_typed\_mem\_get\_info()*, *umask()*

48973 XBD `<fcntl.h>`, `<sys/mman.h>`

**CHANGE HISTORY**

48974 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

**Issue 7**

48977 Austin Group Interpretation 1003.1-2001 #143 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pow()**48978 **NAME**48979 `pow, powf, powl` — power function48980 **SYNOPSIS**48981 `#include <math.h>`48982 `double pow(double x, double y);`48983 `float powf(float x, float y);`48984 `long double powl(long double x, long double y);`48985 **DESCRIPTION**48986 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48987 conflict between the requirements described here and the ISO C standard is unintentional. This  
48988 volume of POSIX.1-2008 defers to the ISO C standard.48989 These functions shall compute the value of  $x$  raised to the power  $y$ ,  $x^y$ . If  $x$  is negative, the  
48990 application shall ensure that  $y$  is an integer value.48991 An application wishing to check for error situations should set *errno* to zero and call  
48992 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
48993 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
48994 zero, an error has occurred.48995 **RETURN VALUE**48996 Upon successful completion, these functions shall return the value of  $x$  raised to the power  $y$ .48997 **MX** For finite values of  $x < 0$ , and finite non-integer values of  $y$ , a domain error shall occur and  
48998 either a NaN (if representable), or an implementation-defined value shall be returned.48999 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and  
49000 *powl()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively, with the  
49001 same sign as the correct value of the function.49002 If the correct value would cause underflow, and is not representable, a range error may occur,  
49003 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.49004 **CX** For  $y < 0$ , if  $x$  is zero, a pole error may occur and *pow()*, *powf()*, and *powl()* shall return  
49005 **MX**  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively. On systems that support the  
49006 IEC 60559 Floating-Point option, a pole error shall occur and *pow()*, *powf()*, and *powl()* shall  
49007 return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively if  $y$  is an odd integer, or  
49008 HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively if  $y$  is not an odd integer.49009 **MX** If  $x$  or  $y$  is a NaN, a NaN shall be returned (unless specified elsewhere in this description).49010 For any value of  $y$  (including NaN), if  $x$  is +1, 1.0 shall be returned.49011 For any value of  $x$  (including NaN), if  $y$  is  $\pm 0$ , 1.0 shall be returned.49012 For any odd integer value of  $y > 0$ , if  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.49013 For  $y > 0$  and not an odd integer, if  $x$  is  $\pm 0$ , +0 shall be returned.49014 If  $x$  is  $-1$ , and  $y$  is  $\pm$ Inf, 1.0 shall be returned.49015 For  $|x| < 1$ , if  $y$  is  $-$ Inf, +Inf shall be returned.49016 For  $|x| > 1$ , if  $y$  is  $-$ Inf, +0 shall be returned.49017 For  $|x| < 1$ , if  $y$  is +Inf, +0 shall be returned.49018 For  $|x| > 1$ , if  $y$  is +Inf, +Inf shall be returned.49019 For  $y$  an odd integer  $< 0$ , if  $x$  is  $-$ Inf,  $-0$  shall be returned.

49020		For $y < 0$ and not an odd integer, if $x$ is $-\text{Inf}$ , $+0$ shall be returned.
49021		For $y$ an odd integer $> 0$ , if $x$ is $-\text{Inf}$ , $-\text{Inf}$ shall be returned.
49022		For $y > 0$ and not an odd integer, if $x$ is $-\text{Inf}$ , $+\text{Inf}$ shall be returned.
49023		For $y < 0$ , if $x$ is $+\text{Inf}$ , $+0$ shall be returned.
49024		For $y > 0$ , if $x$ is $+\text{Inf}$ , $+\text{Inf}$ shall be returned.
49025		If the correct value would cause underflow, and is representable, a range error may occur and
49026		the correct value shall be returned.
49027	<b>ERRORS</b>	
49028		These functions shall fail if:
49029	Domain Error	The value of $x$ is negative and $y$ is a finite non-integer.
49030		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
49031		then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i>
49032		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
49033		shall be raised.
49034	MX Pole Error	The value of $x$ is zero and $y$ is negative.
49035		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
49036		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
49037		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
49038		floating-point exception shall be raised.
49039	Range Error	The result overflows.
49040		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
49041		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
49042		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
49043		floating-point exception shall be raised.
49044		These functions may fail if:
49045	Pole Error	The value of $x$ is zero and $y$ is negative.
49046		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
49047		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
49048		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
49049		floating-point exception shall be raised.
49050	Range Error	The result underflows.
49051		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
49052		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
49053		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
49054		floating-point exception shall be raised.

**pow()**49055 **EXAMPLES**

49056 None.

49057 **APPLICATION USAGE**49058 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
49059 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.49060 **RATIONALE**

49061 None.

49062 **FUTURE DIRECTIONS**

49063 None.

49064 **SEE ALSO**49065 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

49066 XBD Section 4.19 (on page 116), &lt;math.h&gt;

49067 **CHANGE HISTORY**

49068 First released in Issue 1. Derived from Issue 1 of the SVID.

49069 **Issue 5**49070 The DESCRIPTION is updated to indicate how an application should check for an error. This  
49071 text was previously published in the APPLICATION USAGE section.49072 **Issue 6**

49073 The normative text is updated to avoid use of the term “must” for application requirements.

49074 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.49075 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
49076 revised to align with the ISO/IEC 9899:1999 standard.49077 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
49078 marked.49079 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third  
49080 paragraph in the RETURN VALUE section.49081 **Issue 7**

49082 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

*System Interfaces***pread()**49083 **NAME**49084       

```
pread — read from a file
```

49085 **SYNOPSIS**49086       

```
#include <unistd.h>
```

49087       

```
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

49088 **DESCRIPTION**49089       Refer to *read()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**printf()**49090 **NAME**

49091           printf — print formatted output

49092 **SYNOPSIS**

49093           #include &lt;stdio.h&gt;

49094           int printf(const char \*restrict *format*, ...);49095 **DESCRIPTION**49096           Refer to *fprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

49097 **NAME**

49098 pselect, select — synchronous I/O multiplexing

49099 **SYNOPSIS**

```

49100 #include <sys/select.h>
49101 int pselect(int nfd, fd_set *restrict readfds,
49102            fd_set *restrict writefds, fd_set *restrict errorfds,
49103            const struct timespec *restrict timeout,
49104            const sigset_t *restrict sigmask);
49105 int select(int nfd, fd_set *restrict readfds,
49106           fd_set *restrict writefds, fd_set *restrict errorfds,
49107           struct timeval *restrict timeout);
49108 void FD_CLR(int fd, fd_set *fdset);
49109 int FD_ISSET(int fd, fd_set *fdset);
49110 void FD_SET(int fd, fd_set *fdset);
49111 void FD_ZERO(fd_set *fdset);

```

49112 **DESCRIPTION**

49113 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the  
 49114 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for  
 49115 reading, are ready for writing, or have an exceptional condition pending, respectively.

49116 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 49117 • For the *select()* function, the timeout period is given in seconds and microseconds in an  
 49118 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is  
 49119 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 49120 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when  
 49121 *sigmask* is a null pointer.
- 49122 • Upon successful completion, the *select()* function may modify the object pointed to by the  
 49123 *timeout* argument.

49124 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal  
 49125 OB XSR devices, **STREAMS-based files**, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*  
 49126 on file descriptors that refer to other types of file is unspecified.

49127 The *nfd* argument specifies the range of descriptors to be tested. The first *nfd* descriptors shall  
 49128 be checked in each set; that is, the descriptors from zero through *nfd*–1 in the descriptor sets  
 49129 shall be examined.

49130 If the *readfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 49131 specifies the file descriptors to be checked for being ready to read, and on output indicates  
 49132 which file descriptors are ready to read.

49133 If the *writefds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 49134 specifies the file descriptors to be checked for being ready to write, and on output indicates  
 49135 which file descriptors are ready to write.

49136 If the *errorfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 49137 specifies the file descriptors to be checked for error conditions pending, and on output indicates  
 49138 which file descriptors have error conditions pending.

49139 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to  
 49140 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for  
 49141 reading, ready for writing, or have an error condition pending, respectively, and shall return the  
 49142 total number of ready descriptors in all the output sets. For each file descriptor less than *nfd*, the

**pselect()**

49143 corresponding bit shall be set upon successful completion if it was set on input and the  
49144 associated condition is true for that file descriptor.

49145 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*  
49146 function shall block until at least one of the requested operations becomes ready, until the  
49147 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the  
49148 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null  
49149 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified  
49150 time interval expires without any requested operation becoming ready, the function shall return.  
49151 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block  
49152 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*  
49153 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

49154 The use of a timeout does not affect any pending timers set up by *alarm()* or *settimer()*.

49155 Implementations may place limitations on the maximum timeout interval supported. All  
49156 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*  
49157 argument specifies a timeout interval greater than the implementation-defined maximum value,  
49158 the maximum value shall be used as the actual timeout value. Implementations may also place  
49159 limitations on the granularity of timeout intervals. If the requested timeout interval requires a  
49160 finer granularity than the implementation supports, the actual timeout interval shall be rounded  
49161 up to the next supported value.

49162 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the  
49163 caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall  
49164 restore the signal mask of the calling thread before returning.

49165 A descriptor shall be considered ready for reading when a call to an input function with  
49166 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
49167 successfully. (The function might return data, an end-of-file indication, or an error other than  
49168 one indicating that it is blocked, and in each of these cases the descriptor shall be considered  
49169 ready for reading.)

49170 A descriptor shall be considered ready for writing when a call to an output function with  
49171 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
49172 successfully.

49173 If a socket has a pending error, it shall be considered to have an exceptional condition pending.  
49174 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for  
49175 use with a socket, it is protocol-specific except as noted below. For other file types it is  
49176 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or  
49177 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate  
49178 that the descriptor has no exceptional condition pending.

49179 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with  
49180 parameters requesting normal and ancillary data, such that the presence of either type shall  
49181 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if  
49182 the socket option SO\_OOBINLINE has been enabled, as out-of-band data is enqueued with  
49183 normal data. If the socket is currently listening, then it shall be marked as readable if an  
49184 incoming connection request has been received, and a call to the *accept()* function shall complete  
49185 without blocking.

49186 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying  
49187 an amount of normal data equal to the current value of the SO\_SNDLOWAT option for the  
49188 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the  
49189 connection attempt has either succeeded or failed leaving a pending error, the socket shall be

49190 marked as writable.

49191 A socket shall be considered to have an exceptional condition pending if a receive operation  
49192 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would  
49193 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag  
49194 would be used to read out-of-band data.) A socket shall also be considered to have an  
49195 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other  
49196 circumstances under which a socket may be considered to have an exceptional condition  
49197 pending are protocol-specific and implementation-defined.

49198 If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and the `timeout` argument is  
49199 not a null pointer, the `pselect()` or `select()` function shall block for the time specified, or until  
49200 interrupted by a signal. If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and  
49201 the `timeout` argument is a null pointer, the `pselect()` or `select()` function shall block until  
49202 interrupted by a signal.

49203 File descriptors associated with regular files shall always select true for ready to read, ready to  
49204 write, and error conditions.

49205 On failure, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments shall not be  
49206 modified. If the timeout interval expires without the specified condition being true for any of the  
49207 specified file descriptors, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments  
49208 shall have all bits set to 0.

49209 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,  
49210 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a  
49211 macro definition is suppressed in order to access an actual function, or a program defines an  
49212 external identifier with any of these names, the behavior is undefined.

49213 `FD_CLR(fd, fdsetp)` shall remove the file descriptor `fd` from the set pointed to by `fdsetp`. If `fd` is not  
49214 a member of this set, there shall be no effect on the set, nor will an error be returned.

49215 `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor `fd` is a member of the set  
49216 pointed to by `fdsetp`, and shall evaluate to zero otherwise.

49217 `FD_SET(fd, fdsetp)` shall add the file descriptor `fd` to the set pointed to by `fdsetp`. If the file  
49218 descriptor `fd` is already in this set, there shall be no effect on the set, nor will an error be  
49219 returned.

49220 `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by `fdsetp` to the null set. No error is  
49221 returned if the set is not empty at the time `FD_ZERO()` is invoked.

49222 The behavior of these macros is undefined if the `fd` argument is less than 0 or greater than or  
49223 equal to `FD_SETSIZE`, or if `fd` is not a valid file descriptor, or if any of the arguments are  
49224 expressions with side-effects.

49225 If a thread gets canceled during a `pselect()` call, the signal mask in effect when executing the  
49226 registered cleanup functions is either the original signal mask or the signal mask installed as part  
49227 of the `pselect()` call.

#### 49228 RETURN VALUE

49229 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of  
49230 bits set in the bit masks. Otherwise, `-1` shall be returned, and `errno` shall be set to indicate the  
49231 error.

49232 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` do not return a value. `FD_ISSET()` shall return a non-  
49233 zero value if the bit for the file descriptor `fd` is set in the file descriptor set pointed to by `fdset`, and  
49234 0 otherwise.

**pselect()**49235 **ERRORS**

49236 Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:

- 49237 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a  
49238 valid open file descriptor.
- 49239 [EINTR] The function was interrupted before any of the selected events occurred and  
49240 before the timeout interval expired.
- 49241 XSI If SA\_RESTART has been set for the interrupting signal, it is implementation-  
49242 defined whether the function restarts or returns with [EINTR].
- 49243 [EINVAL] An invalid timeout interval was specified.
- 49244 [EINVAL] The *nfds* argument is less than 0 or greater than FD\_SETSIZE.
- 49245 OB XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is  
49246 linked (directly or indirectly) downstream from a multiplexer.

49247 **EXAMPLES**

49248 None.

49249 **APPLICATION USAGE**

49250 None.

49251 **RATIONALE**

49252 In earlier versions of the Single UNIX Specification, the *select()* function was defined in the  
49253 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was  
49254 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the  
49255 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.  
49256 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`  
49257 to include `<sys/select.h>`.

49258 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers  
49259 can install an additional cancellation handler which resets the signal mask to the expected value.

```

49260 void cleanup(void *arg)
49261 {
49262     sigset_t *ss = (sigset_t *) arg;
49263     pthread_sigmask(SIG_SETMASK, ss, NULL);
49264 }
49265 int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
49266                fd_set errorfds, const struct timespec *timeout,
49267                const sigset_t *sigmask)
49268 {
49269     sigset_t oldmask;
49270     int result;
49271     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
49272     pthread_cleanup_push(cleanup, &oldmask);
49273     result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
49274     pthread_cleanup_pop(0);
49275     return result;
49276 }

```

49277 **FUTURE DIRECTIONS**

49278 None.

49279 **SEE ALSO**49280 *accept()*, *alarm()*, *connect()*, *fcntl()*, *getitimer()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *write()*49281 XBD [<sys/select.h>](#), [<sys/time.h>](#)49282 **CHANGE HISTORY**

49283 First released in Issue 4, Version 2.

49284 **Issue 5**

49285 Moved from X/OPEN UNIX extension to BASE.

49286 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when  
 49287 *nfds* is less than 0 or greater than FD\_SETSIZE. It previously stated less than 0, or greater than or  
 49288 equal to FD\_SETSIZE.

49289 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.49290 **Issue 6**49291 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*  
 49292 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

49293 Text referring to sockets is added to the DESCRIPTION.

49294 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 49295 marked as part of the XSI STREAMS Option Group.49296 The following new requirements on POSIX implementations derive from alignment with the  
 49297 Single UNIX Specification:

- 49298
- These functions are now mandatory.

49299 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail  
 49300 related to sockets semantics is added to the DESCRIPTION.49301 The *select()* function now requires inclusion of [<sys/select.h>](#).49302 The **restrict** keyword is added to the *select()* prototype for alignment with the  
 49303 ISO/IEC 9899:1999 standard.49304 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the  
 49305 DESCRIPTION to reference the signal mask in terms of the calling thread rather than the  
 49306 process.49307 **Issue 7**49308 SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled  
 49309 during a call to *pselect()*, and adding example code to the RATIONALE.

49310 Functionality relating to the XSI STREAMS option is marked obsolescent.

49311 Functionality relating to the Threads option is moved to the Base.

**psiginfo()**49312 **NAME**

49313           psiginfo, psignal — print signal information to standard error

49314 **SYNOPSIS**

```
49315 CX       #include <signal.h>
49316       void psiginfo(const siginfo_t *pinfo, const char *message);
49317       void psignal(int signum, const char *message);
```

49318 **DESCRIPTION**

49319       The *psiginfo()* and *psignal()* functions shall print a message out on *stderr* associated with a signal  
49320       number. If *message* is not null and is not the empty string, then the string pointed to by the  
49321       *message* argument shall be printed first, followed by a <colon>, a <space>, and the signal  
49322       description string indicated by *signum*, or by the signal associated with *pinfo*. If the *message*  
49323       argument is null or points to an empty string, then only the signal description shall be printed.  
49324       For *psiginfo()*, the argument *pinfo* references a valid **siginfo\_t** structure. For *psignal()*, if *signum* is  
49325       not a valid signal number, the behavior is implementation-defined.

49326       The *psiginfo()* and *psignal()* functions shall not change the orientation of the standard error  
49327       stream.

49328       The *psiginfo()* and *psignal()* functions shall mark for update the last data modification and last  
49329       file status change timestamps of the file associated with the standard error stream at some time  
49330       between their successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()*  
49331       on *stderr*.

49332 **RETURN VALUE**

49333       These functions shall not return a value.

49334 **ERRORS**

49335       No errors are defined.

49336 **EXAMPLES**

49337       None.

49338 **APPLICATION USAGE**

49339       None.

49340 **RATIONALE**

49341       System V historically has *psignal()* and *psiginfo()* in <**siginfo.h**>. However, the <**siginfo.h**>  
49342       header is not specified in the Base Definitions volume of POSIX.1-2008, and the type **siginfo\_t** is  
49343       defined in <**signal.h**>.

49344 **FUTURE DIRECTIONS**

49345       None.

49346 **SEE ALSO**49347       *perorr()*, *strsignal()*49348       XBD <**signal.h**>49349 **CHANGE HISTORY**

49350       First released in Issue 7.

49351 **NAME**

49352 pthread\_atfork — register fork handlers

49353 **SYNOPSIS**

49354 #include &lt;pthread.h&gt;

49355 int pthread\_atfork(void (\*prepare)(void), void (\*parent)(void),  
49356 void (\*child)(void));49357 **DESCRIPTION**

49358 The *pthread\_atfork()* function shall declare fork handlers to be called before and after *fork()*, in  
 49359 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*  
 49360 processing commences. The *parent* fork handle shall be called after *fork()* processing completes  
 49361 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in  
 49362 the child process. If no handling is desired at one or more of these three points, the  
 49363 corresponding fork handler address(es) may be set to NULL.

49364 The order of calls to *pthread\_atfork()* is significant. The *parent* and *child* fork handlers shall be  
 49365 called in the order in which they were established by calls to *pthread\_atfork()*. The *prepare* fork  
 49366 handlers shall be called in the opposite order.

49367 **RETURN VALUE**

49368 Upon successful completion, *pthread\_atfork()* shall return a value of zero; otherwise, an error  
 49369 number shall be returned to indicate the error.

49370 **ERRORS**49371 The *pthread\_atfork()* function shall fail if:

49372 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

49373 The *pthread\_atfork()* function shall not return an error code of [EINTR].49374 **EXAMPLES**

49375 None.

49376 **APPLICATION USAGE**

49377 None.

49378 **RATIONALE**

49379 There are at least two serious problems with the semantics of *fork()* in a multi-threaded  
 49380 program. One problem has to do with state (for example, memory) covered by mutexes.  
 49381 Consider the case where one thread has a mutex locked and the state covered by that mutex is  
 49382 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state  
 49383 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply  
 49384 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about  
 49385 how to correct or otherwise deal with the inconsistent state in the child.

49386 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the  
 49387 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe  
 49388 library routines are promised to be available.

49389 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application  
 49390 programs may not be aware that a multi-threaded library is in use, and they feel free to call any  
 49391 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,  
 49392 they may be extant single-threaded programs and cannot, therefore, be expected to obey new  
 49393 restrictions imposed by the threads library.

49394 On the other hand, the multi-threaded library needs a way to protect its internal state during  
 49395 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-

**pthread\_atfork()**

49396 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to  
49397 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may  
49398 entail simply resetting the state in the child after the *fork()* processing completes.

49399 The *pthread\_atfork()* function was intended to provide multi-threaded libraries with a means to  
49400 protect themselves from innocent application programs that call *fork()*, and to provide multi-  
49401 threaded application programs with a standard mechanism for protecting themselves from  
49402 *fork()* calls in a library routine or the application itself.

49403 The expected usage was that the prepare handler would acquire all mutex locks and the other  
49404 two fork handlers would release them.

49405 For example, an application could have supplied a prepare routine that acquires the necessary  
49406 mutexes the library maintains and supplied child and parent routines that release those  
49407 mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the  
49408 library (and that no mutexes would have been left stranded). This is good in theory, but in  
49409 reality not practical. Each and every mutex and lock in the process must be located and locked.  
49410 Every component of a program including third-party components must participate and they  
49411 must agree who is responsible for which mutex or lock. This is especially problematic for  
49412 mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the  
49413 implementation must be locked, too. This possibly delays the thread calling *fork()* for a long  
49414 time or even indefinitely since uses of these synchronization objects may not be under control of  
49415 the application. A final problem to mention here is the problem of locking streams. At least the  
49416 streams under control of the system (like *stdin*, *stdout*, *stderr*) must be protected by locking the  
49417 stream with *flockfile()*. But the application itself could have done that, possibly in the same  
49418 thread calling *fork()*. In this case, the process will deadlock.

49419 Alternatively, some libraries might have been able to supply just a *child* routine that reinitializes  
49420 the mutexes in the library and all associated states to some known value (for example, what it  
49421 was when the image was originally executed). This approach is not possible, though, because  
49422 implementations are allowed to fail *\*\_init()* and *\*\_destroy()* calls for mutexes and locks if the  
49423 mutex or lock is still locked. In this case, the *child* routine is not able to reinitialize the mutexes  
49424 and locks.

49425 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization  
49426 variables remain in the same state in the child as they were in the parent at the time *fork()* was  
49427 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child  
49428 process, and any associated states may be inconsistent. The intention was that the parent process  
49429 could have avoided this by explicit code that acquires and releases locks critical to the child via  
49430 *pthread\_atfork()*. In addition, any critical threads would have needed to be recreated and  
49431 reinitialized to the proper state in the child (also via *pthread\_atfork()*).

49432 A higher-level package may acquire locks on its own data structures before invoking lower-level  
49433 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of  
49434 initialization for avoiding package deadlock: a package initializes all packages on which it  
49435 depends before it calls the *pthread\_atfork()* function for itself.

49436 As explained, there is no suitable solution for functionality which requires non-atomic  
49437 operations to be protected through mutexes and locks. This is why the POSIX.1 standard since  
49438 the 1996 release requires that the child process after *fork()* in a multi-threaded process only calls  
49439 *async-signal-safe* interfaces.

49440 **FUTURE DIRECTIONS**

49441 None.

49442 **SEE ALSO**49443 *atexit()*, *exec*, *fork()*49444 XBD `<pthread.h>`, `<sys/types.h>`49445 **CHANGE HISTORY**

49446 First released in Issue 5. Derived from the POSIX Threads Extension.

49447 IEEE PASC Interpretation 1003.1c #4 is applied.

49448 **Issue 6**49449 The *pthread\_atfork()* function is marked as part of the Threads option.49450 The `<pthread.h>` header is added to the SYNOPSIS.49451 **Issue 7**49452 The *pthread\_atfork()* function is moved from the Threads option to the Base.

49453 SD5-XSH-ERN-145 is applied, updating the RATIONALE.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_attr\_destroy()**

System Interfaces

49454 **NAME**

49455 pthread\_attr\_destroy, pthread\_attr\_init — destroy and initialize the thread attributes object

49456 **SYNOPSIS**

```
49457 #include <pthread.h>
49458 int pthread_attr_destroy(pthread_attr_t *attr);
49459 int pthread_attr_init(pthread_attr_t *attr);
```

49460 **DESCRIPTION**

49461 The *pthread\_attr\_destroy()* function shall destroy a thread attributes object. An implementation  
 49462 may cause *pthread\_attr\_destroy()* to set *attr* to an implementation-defined invalid value. A  
 49463 destroyed *attr* attributes object can be reinitialized using *pthread\_attr\_init()*; the results of  
 49464 otherwise referencing the object after it has been destroyed are undefined.

49465 The *pthread\_attr\_init()* function shall initialize a thread attributes object *attr* with the default  
 49466 value for all of the individual attributes used by a given implementation.

49467 The resulting attributes object (possibly modified by setting individual attribute values) when  
 49468 used by *pthread\_create()* defines the attributes of the thread created. A single attributes object can  
 49469 be used in multiple simultaneous calls to *pthread\_create()*. Results are undefined if  
 49470 *pthread\_attr\_init()* is called specifying an already initialized *attr* attributes object.

49471 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_destroy()*  
 49472 does not refer to an initialized thread attributes object.

49473 **RETURN VALUE**

49474 Upon successful completion, *pthread\_attr\_destroy()* and *pthread\_attr\_init()* shall return a value of  
 49475 0; otherwise, an error number shall be returned to indicate the error.

49476 **ERRORS**

49477 The *pthread\_attr\_init()* function shall fail if:

49478 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

49479 These functions shall not return an error code of [EINTR].

49480 **EXAMPLES**

49481 None.

49482 **APPLICATION USAGE**

49483 None.

49484 **RATIONALE**

49485 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
 49486 support probable future standardization in these areas without requiring that the function itself  
 49487 be changed.

49488 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
 49489 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
 49490 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
 49491 can help by allowing the changes to be isolated in a single place, rather than being spread across  
 49492 every instance of thread creation.

49493 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,  
 49494 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
 49495 can be defined in a single place and then referenced wherever threads need to be created.  
 49496 Changes to “class” decisions become straightforward, and detailed analysis of each  
 49497 *pthread\_create()* call is not required.

49498 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
 49499 been specified as structures, adding new attributes would force recompilation of all multi-  
 49500 threaded programs when the attributes objects are extended; this might not be possible if  
 49501 different program components were supplied by different vendors.

49502 Additionally, opaque attributes objects present opportunities for improving performance.  
 49503 Argument validity can be checked once when attributes are set, rather than each time a thread is  
 49504 created. Implementations often need to cache kernel objects that are expensive to create.  
 49505 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
 49506 invalid due to attribute changes.

49507 Since assignment is not necessarily defined on a given opaque type, implementation-defined  
 49508 default values cannot be defined in a portable way. The solution to this problem is to allow  
 49509 attributes objects to be initialized dynamically by attributes object initialization functions, so that  
 49510 default values can be supplied automatically by the implementation.

49511 The following proposal was provided as a suggested alternative to the supplied attributes:

- 49512 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
 49513 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
 49514 parameter containing the flags should be an opaque type for extensibility. If no flags are  
 49515 set in the parameter, then the objects are created with default characteristics. An  
 49516 implementation may specify implementation-defined flag values and associated  
 49517 behavior.
- 49518 2. If further specialization of mutexes and condition variables is necessary, implementations  
 49519 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
 49520 **pthread\_cond\_t** objects (instead of on attributes objects).

49521 The difficulties with this solution are:

- 49522 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using  
 49523 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
 49524 application programmers need to know the location of each bit. If bits are set or read by  
 49525 encapsulation (that is, get and set functions), then the bitmask is merely an  
 49526 implementation of attributes objects as currently defined and should not be exposed to  
 49527 the programmer.
- 49528 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
 49529 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking  
 49530 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,  
 49531 the bitmask can only reasonably control whether particular attributes are set or not, and it  
 49532 cannot serve as the repository of the value itself. The value needs to be specified as a  
 49533 function parameter (which is non-extensible), or by setting a structure field (which is non-  
 49534 opaque), or by get and set functions (making the bitmask a redundant addition to the  
 49535 attributes objects).

49536 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
 49537 machine-dependent. Some implementations may not be able to change the size of the stack, for  
 49538 example, and others may not need to because stack pages may be discontinuous and can be  
 49539 allocated and released on demand.

49540 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
 49541 to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2008 has  
 49542 to be done with care so as not to affect binary-compatibility.

49543 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,

**pthread\_attr\_destroy()**

49544 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
49545 implementation with extensions to **pthread\_attr\_t** cannot look beyond the area that the binary  
49546 application assumes is valid. This suggests that implementations should maintain a size field in  
49547 the attributes object, as well as possibly version information, if extensions in different directions  
49548 (possibly by different vendors) are to be accommodated.

49549 If an implementation detects that the value specified by the *attr* argument to  
49550 *pthread\_attr\_destroy()* does not refer to an initialized thread attributes object, it is recommended  
49551 that the function should fail and report an [EINVAL] error.

49552 If an implementation detects that the value specified by the *attr* argument to *pthread\_attr\_init()*  
49553 refers to an already initialized thread attributes object, it is recommended that the function  
49554 should fail and report an [EBUSY] error.

**FUTURE DIRECTIONS**

49555 None.

**SEE ALSO**

49558 *pthread\_attr\_getstacksize()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*

49559 XBD <pthread.h>

**CHANGE HISTORY**

49560 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

49563 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions are marked as part of the Threads  
49564 option.

49565 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already  
49566 initialized thread attributes object is undefined.

49567 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS  
49568 section to add the optional [EINVAL] error for the *pthread\_attr\_destroy()* function, and the  
49569 optional [EBUSY] error for the *pthread\_attr\_init()* function.

**Issue 7**

49570 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions are moved from the Threads option  
49571 to the Base.

49573 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49574 results in undefined behavior.

49575 The [EBUSY] error for an already initialized thread attributes object is removed; this condition  
49576 results in undefined behavior.

49577 **NAME**

49578 pthread\_attr\_getdetachstate, pthread\_attr\_setdetachstate — get and set the detachstate attribute

49579 **SYNOPSIS**

49580 #include &lt;pthread.h&gt;

49581 int pthread\_attr\_getdetachstate(const pthread\_attr\_t \*attr,  
49582 int \*detachstate);

49583 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

49584 **DESCRIPTION**49585 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread  
49586 is created detached, then use of the ID of the newly created thread by the *pthread\_detach()* or  
49587 *pthread\_join()* function is an error.49588 The *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* functions, respectively, shall get  
49589 and set the *detachstate* attribute in the *attr* object.49590 For *pthread\_attr\_getdetachstate()*, *detachstate* shall be set to either  
49591 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.49592 For *pthread\_attr\_setdetachstate()*, the application shall set *detachstate* to either  
49593 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.49594 A value of PTHREAD\_CREATE\_DETACHED shall cause all threads created with *attr* to be in  
49595 the detached state, whereas using a value of PTHREAD\_CREATE\_JOINABLE shall cause all  
49596 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute  
49597 shall be PTHREAD\_CREATE\_JOINABLE.49598 The behavior is undefined if the value specified by the *attr* argument to  
49599 *pthread\_attr\_getdetachstate()* or *pthread\_attr\_setdetachstate()* does not refer to an initialized thread  
49600 attributes object.49601 **RETURN VALUE**49602 Upon successful completion, *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* shall  
49603 return a value of 0; otherwise, an error number shall be returned to indicate the error.49604 The *pthread\_attr\_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*  
49605 if successful.49606 **ERRORS**49607 The *pthread\_attr\_setdetachstate()* function shall fail if:49608 [EINVAL] The value of *detachstate* was not valid

49609 These functions shall not return an error code of [EINTR].

49610 **EXAMPLES**49611 **Retrieving the detachstate Attribute**49612 This example shows how to obtain the *detachstate* attribute of a thread attribute object.

49613 #include &lt;pthread.h&gt;

49614 pthread\_attr\_t thread\_attr;

49615 int detachstate;

49616 int rc;

49617 /\* code initializing thread\_attr \*/

49618 ...

**pthread\_attr\_getdetachstate()**

```

49619     rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
49620     if (rc!=0) {
49621         /* handle error */
49622         ...
49623     }
49624     else {
49625         /* legal values for detachstate are:
49626          * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
49627          */
49628         ...
49629     }

```

**49630 APPLICATION USAGE**

49631 None.

**49632 RATIONALE**

49633 If an implementation detects that the value specified by the *attr* argument to  
49634 *pthread\_attr\_getdetachstate()* or *pthread\_attr\_setdetachstate()* does not refer to an initialized thread  
49635 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

**49636 FUTURE DIRECTIONS**

49637 None.

**49638 SEE ALSO**

49639 [pthread\\_attr\\_destroy\(\)](#), [pthread\\_attr\\_getstacksize\(\)](#), [pthread\\_create\(\)](#)

49640 XBD <[pthread.h](#)>

**49641 CHANGE HISTORY**

49642 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**49643 Issue 6**

49644 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are marked as part of  
49645 the Threads option.

49646 The normative text is updated to avoid use of the term “must” for application requirements.

49647 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the  
49648 EXAMPLES section.

49649 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS  
49650 section to include the optional [EINVAL] error.

**49651 Issue 7**

49652 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are moved from the  
49653 Threads option to the Base.

49654 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49655 results in undefined behavior.

49656 **NAME**

49657 pthread\_attr\_getguardsize, pthread\_attr\_setguardsize — get and set the thread guardsize  
 49658 attribute

49659 **SYNOPSIS**

```
49660 #include <pthread.h>
49661 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
49662 size_t *restrict guardsize);
49663 int pthread_attr_setguardsize(pthread_attr_t *attr,
49664 size_t guardsize);
```

49665 **DESCRIPTION**

49666 The *pthread\_attr\_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This  
 49667 attribute shall be returned in the *guardsize* parameter.

49668 The *pthread\_attr\_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new  
 49669 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard  
 49670 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard  
 49671 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

49672 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The  
 49673 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is  
 49674 created with guard protection, the implementation allocates extra memory at the overflow end  
 49675 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows  
 49676 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

49677 A conforming implementation may round up the value contained in *guardsize* to a multiple of  
 49678 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation  
 49679 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread\_attr\_getguardsize()*  
 49680 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous  
 49681 *pthread\_attr\_setguardsize()* function call.

49682 The default value of the *guardsize* attribute is implementation-defined.

49683 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread  
 49684 stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the  
 49685 implementation. It is the responsibility of the application to manage stack overflow along with  
 49686 stack allocation and management in this case.

49687 The behavior is undefined if the value specified by the *attr* argument to  
 49688 *pthread\_attr\_getguardsize()* or *pthread\_attr\_setguardsize()* does not refer to an initialized thread  
 49689 attributes object.

49690 **RETURN VALUE**

49691 If successful, the *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall return  
 49692 zero; otherwise, an error number shall be returned to indicate the error.

49693 **ERRORS**

49694 These functions shall fail if:

49695 [EINVAL] The parameter *guardsize* is invalid.

49696 These functions shall not return an error code of [EINTR].

**pthread\_attr\_getguardsize()**49697 **EXAMPLES**49698 **Retrieving the guardsize Attribute**

49699 This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```

49700 #include <pthread.h>
49701 pthread_attr_t thread_attr;
49702 size_t guardsize;
49703 int rc;
49704 /* code initializing thread_attr */
49705 ...
49706 rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
49707 if (rc != 0) {
49708     /* handle error */
49709     ...
49710 }
49711 else {
49712     if (guardsize > 0) {
49713         /* a guard area of at least guardsize bytes is provided */
49714         ...
49715     }
49716     else {
49717         /* no guard area provided */
49718         ...
49719     }
49720 }

```

49721 **APPLICATION USAGE**

49722 None.

49723 **RATIONALE**

49724 The *guardsize* attribute is provided to the application for two reasons:

- 49725 1. Overflow protection can potentially result in wasted system resources. An application  
49726 that creates a large number of threads, and which knows its threads never overflow their  
49727 stack, can save system resources by turning off guard areas.
- 49728 2. When threads allocate large data structures on the stack, large guard areas may be needed  
49729 to detect stack overflow.

49730 The default size of the guard area is left implementation-defined since on systems supporting  
49731 very large page sizes, the overhead might be substantial if at least one guard page is required by  
49732 default.

49733 If an implementation detects that the value specified by the *attr* argument to  
49734 *pthread\_attr\_getguardsize()* or *pthread\_attr\_setguardsize()* does not refer to an initialized thread  
49735 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

49736 **FUTURE DIRECTIONS**

49737 None.

49738 **SEE ALSO**49739 XBD [<pthread.h>](#), [<sys/mman.h>](#)49740 **CHANGE HISTORY**

49741 First released in Issue 5.

49742 **Issue 6**49743 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the  
49744 second error condition.49745 The **restrict** keyword is added to the *pthread\_attr\_getguardsize()* prototype for alignment with the  
49746 ISO/IEC 9899:1999 standard.49747 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS  
49748 section to remove the [EINVAL] error (“The attribute *attr* is invalid.”), and replacing it with the  
49749 optional [EINVAL] error.49750 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the  
49751 EXAMPLES section.49752 **Issue 7**49753 SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.49754 SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the  
49755 guard area is implementation-defined.49756 The *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions are moved from the XSI  
49757 option to the Base.49758 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49759 results in undefined behavior.

**pthread\_attr\_getinheritsched()**49760 **NAME**

49761 pthread\_attr\_getinheritsched, pthread\_attr\_setinheritsched — get and set the inheritsched  
 49762 attribute (**REALTIME THREADS**)

49763 **SYNOPSIS**

```
49764 TPS #include <pthread.h>
49765 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
49766 int *restrict inheritsched);
49767 int pthread_attr_setinheritsched(pthread_attr_t *attr,
49768 int inheritsched);
```

49769 **DESCRIPTION**

49770 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions, respectively, shall  
 49771 get and set the *inheritsched* attribute in the *attr* argument.

49772 When the attributes objects are used by *pthread\_create()*, the *inheritsched* attribute determines  
 49773 how the other scheduling attributes of the created thread shall be set.

49774 The supported values of *inheritsched* shall be:

49775 **PTHREAD\_INHERIT\_SCHED**

49776 Specifies that the thread scheduling attributes shall be inherited from the creating thread,  
 49777 and the scheduling attributes in this *attr* argument shall be ignored.

49778 **PTHREAD\_EXPLICIT\_SCHED**

49779 Specifies that the thread scheduling attributes shall be set to the corresponding values from  
 49780 this attributes object.

49781 The symbols **PTHREAD\_INHERIT\_SCHED** and **PTHREAD\_EXPLICIT\_SCHED** are defined in  
 49782 the **<pthread.h>** header.

49783 The following thread scheduling attributes defined by POSIX.1-2008 are affected by the  
 49784 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and  
 49785 scheduling contention scope (*contentionscope*).

49786 The behavior is undefined if the value specified by the *attr* argument to  
 49787 *pthread\_attr\_getinheritsched()* or *pthread\_attr\_setinheritsched()* does not refer to an initialized  
 49788 thread attributes object.

49789 **RETURN VALUE**

49790 If successful, the *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions shall  
 49791 return zero; otherwise, an error number shall be returned to indicate the error.

49792 **ERRORS**

49793 The *pthread\_attr\_setinheritsched()* function may fail if:

49794 [EINVAL] The value of *inheritsched* is not valid.

49795 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49796 These functions shall not return an error code of [EINTR].

49797 **EXAMPLES**

49798 None.

49799 **APPLICATION USAGE**49800 After these attributes have been set, a thread can be created with the specified attributes using  
49801 *pthread\_create()*. Using these routines does not affect the current running thread.49802 See [Section 2.9.4](#) (on page 509) for further details on thread scheduling attributes and their  
49803 default settings.49804 **RATIONALE**49805 If an implementation detects that the value specified by the *attr* argument to  
49806 *pthread\_attr\_getinheritsched()* or *pthread\_attr\_setinheritsched()* does not refer to an initialized  
49807 thread attributes object, it is recommended that the function should fail and report an [EINVAL]  
49808 error.49809 **FUTURE DIRECTIONS**

49810 None.

49811 **SEE ALSO**49812 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getschedpolicy()*,  
49813 *pthread\_attr\_getschedparam()*, *pthread\_create()*49814 XBD [<pthread.h>](#), [<sched.h>](#)49815 **CHANGE HISTORY**

49816 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49817 Marked as part of the Realtime Threads Feature Group.

49818 **Issue 6**49819 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked as part  
49820 of the Threads and Thread Execution Scheduling options.49821 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
49822 implementation does not support the Thread Execution Scheduling option.49823 The **restrict** keyword is added to the *pthread\_attr\_getinheritsched()* prototype for alignment with  
49824 the ISO/IEC 9899:1999 standard.49825 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of  
49826 *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS  
49827 section for checking when *attr* refers to an uninitialized thread attribute object.49828 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to  
49829 [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.49830 **Issue 7**49831 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are moved from the  
49832 Threads option.49833 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49834 results in undefined behavior.

**pthread\_attr\_getschedparam()**

System Interfaces

49835 **NAME**

49836 pthread\_attr\_getschedparam, pthread\_attr\_setschedparam — get and set the schedparam  
49837 attribute

49838 **SYNOPSIS**

```
49839 #include <pthread.h>
49840 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
49841 struct sched_param *restrict param);
49842 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
49843 const struct sched_param *restrict param);
```

49844 **DESCRIPTION**

49845 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions, respectively, shall  
49846 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*  
49847 structure are defined in the **<sched.h>** header. For the SCHED\_FIFO and SCHED\_RR policies,  
49848 the only required member of *param* is *sched\_priority*.

49849 TSP For the SCHED\_SPORADIC policy, the required members of the *param* structure are  
49850 *sched\_priority*, *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and  
49851 *sched\_ss\_max\_repl*. The specified *sched\_ss\_repl\_period* must be greater than or equal to the  
49852 specified *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.  
49853 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
49854 function to succeed; if not, the function shall fail. It is unspecified whether the  
49855 *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are stored as provided by this function or are  
49856 rounded to align with the resolution of the clock being used.

49857 The behavior is undefined if the value specified by the *attr* argument to  
49858 *pthread\_attr\_getschedparam()* or *pthread\_attr\_setschedparam()* does not refer to an initialized thread  
49859 attributes object.

49860 **RETURN VALUE**

49861 If successful, the *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions shall  
49862 return zero; otherwise, an error number shall be returned to indicate the error.

49863 **ERRORS**

49864 The *pthread\_attr\_setschedparam()* function may fail if:

- 49865 [EINVAL] The value of *param* is not valid.
- 49866 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.
- 49867 These functions shall not return an error code of [EINTR].

49868 **EXAMPLES**

49869 None.

49870 **APPLICATION USAGE**

49871 After these attributes have been set, a thread can be created with the specified attributes using  
49872 *pthread\_create()*. Using these routines does not affect the current running thread.

49873 **RATIONALE**

49874 If an implementation detects that the value specified by the *attr* argument to  
49875 *pthread\_attr\_getschedparam()* or *pthread\_attr\_setschedparam()* does not refer to an initialized thread  
49876 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

**49877 FUTURE DIRECTIONS**

49878 None.

**49879 SEE ALSO**

49880 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
49881 *pthread\_attr\_getschedpolicy()*, *pthread\_create()*

49882 XBD **<pthread.h>**, **<sched.h>**

**49883 CHANGE HISTORY**

49884 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**49885 Issue 6**

49886 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are marked as part  
49887 of the Threads option.

49888 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

49889 The **restrict** keyword is added to the *pthread\_attr\_getschedparam()* and  
49890 *pthread\_attr\_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

49891 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS  
49892 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
49893 object.

**49894 Issue 7**

49895 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are moved from the  
49896 Threads option to the Base.

49897 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
49898 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.

49899 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49900 results in undefined behavior.

**pthread\_attr\_getschedpolicy()**

System Interfaces

49901 **NAME**

49902 pthread\_attr\_getschedpolicy, pthread\_attr\_setschedpolicy — get and set the schedpolicy  
 49903 attribute (**REALTIME THREADS**)

49904 **SYNOPSIS**

```
49905 TPS #include <pthread.h>
49906
49907 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
49908                               int *restrict policy);
49908 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

49909 **DESCRIPTION**

49910 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions, respectively, shall  
 49911 get and set the *schedpolicy* attribute in the *attr* argument.

49912 The supported values of *policy* shall include SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER,  
 49913 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy  
 49914 TSP SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC are waiting on a mutex, they shall acquire  
 49915 the mutex in priority order when the mutex is unlocked.

49916 The behavior is undefined if the value specified by the *attr* argument to  
 49917 *pthread\_attr\_getschedpolicy()* or *pthread\_attr\_setschedpolicy()* does not refer to an initialized thread  
 49918 attributes object.

49919 **RETURN VALUE**

49920 If successful, the *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions shall  
 49921 return zero; otherwise, an error number shall be returned to indicate the error.

49922 **ERRORS**

49923 The *pthread\_attr\_setschedpolicy()* function may fail if:

49924 [EINVAL] The value of *policy* is not valid.

49925 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49926 These functions shall not return an error code of [EINTR].

49927 **EXAMPLES**

49928 None.

49929 **APPLICATION USAGE**

49930 After these attributes have been set, a thread can be created with the specified attributes using  
 49931 *pthread\_create()*. Using these routines does not affect the current running thread.

49932 See Section 2.9.4 (on page 509) for further details on thread scheduling attributes and their  
 49933 default settings.

49934 **RATIONALE**

49935 If an implementation detects that the value specified by the *attr* argument to  
 49936 *pthread\_attr\_getschedpolicy()* or *pthread\_attr\_setschedpolicy()* does not refer to an initialized thread  
 49937 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

49938 **FUTURE DIRECTIONS**

49939 None.

49940 **SEE ALSO**

49941 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
 49942 *pthread\_attr\_getschedparam()*, *pthread\_create()*

49943 XBD **<pthread.h>**, **<sched.h>**

**49944 CHANGE HISTORY**

49945 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49946 Marked as part of the Realtime Threads Feature Group.

**49947 Issue 6**

49948 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked as part of  
49949 the Threads and Thread Execution Scheduling options.

49950 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
49951 implementation does not support the Thread Execution Scheduling option.

49952 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

49953 The **restrict** keyword is added to the *pthread\_attr\_getschedpolicy()* prototype for alignment with  
49954 the ISO/IEC 9899:1999 standard.

49955 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to  
49956 [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.

49957 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS  
49958 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
49959 object.

**49960 Issue 7**

49961 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are moved from the  
49962 Threads option.

49963 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
49964 results in undefined behavior.

**pthread\_attr\_getscope()**

System Interfaces

49965 **NAME**

49966 pthread\_attr\_getscope, pthread\_attr\_setscope — get and set the contentionscope attribute  
 49967 (REALTIME THREADS)

49968 **SYNOPSIS**

```
49969 TPS #include <pthread.h>
49970 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
49971 int *restrict contentionscope);
49972 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

49973 **DESCRIPTION**

49974 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions, respectively, shall get and set  
 49975 the *contentionscope* attribute in the *attr* object.

49976 The *contentionscope* attribute may have the values PTHREAD\_SCOPE\_SYSTEM, signifying  
 49977 system scheduling contention scope, or PTHREAD\_SCOPE\_PROCESS, signifying process  
 49978 scheduling contention scope. The symbols PTHREAD\_SCOPE\_SYSTEM and  
 49979 PTHREAD\_SCOPE\_PROCESS are defined in the **<pthread.h>** header.

49980 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_getscope()* or  
 49981 *pthread\_attr\_setscope()* does not refer to an initialized thread attributes object.

49982 **RETURN VALUE**

49983 If successful, the *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions shall return zero;  
 49984 otherwise, an error number shall be returned to indicate the error.

49985 **ERRORS**

49986 The *pthread\_attr\_setscope()* function may fail if:

49987 [EINVAL] The value of *contentionscope* is not valid.

49988 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49989 These functions shall not return an error code of [EINTR].

49990 **EXAMPLES**

49991 None.

49992 **APPLICATION USAGE**

49993 After these attributes have been set, a thread can be created with the specified attributes using  
 49994 *pthread\_create()*. Using these routines does not affect the current running thread.

49995 See Section 2.9.4 (on page 509) for further details on thread scheduling attributes and their  
 49996 default settings.

49997 **RATIONALE**

49998 If an implementation detects that the value specified by the *attr* argument to  
 49999 *pthread\_attr\_getscope()* or *pthread\_attr\_setscope()* does not refer to an initialized thread attributes  
 50000 object, it is recommended that the function should fail and report an [EINVAL] error.

50001 **FUTURE DIRECTIONS**

50002 None.

50003 **SEE ALSO**

50004 *pthread\_attr\_destroy()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedpolicy()*,  
 50005 *pthread\_attr\_getschedparam()*, *pthread\_create()*

50006 XBD **<pthread.h>**, **<sched.h>**

50007 **CHANGE HISTORY**

50008 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50009 Marked as part of the Realtime Threads Feature Group.

50010 **Issue 6**

50011 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked as part of the  
50012 Threads and Thread Execution Scheduling options.

50013 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
50014 implementation does not support the Thread Execution Scheduling option.

50015 The **restrict** keyword is added to the *pthread\_attr\_getscope()* prototype for alignment with the  
50016 ISO/IEC 9899: 1999 standard.

50017 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to  
50018 [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.

50019 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS  
50020 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
50021 object.

50022 **Issue 7**

50023 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are moved from the Threads  
50024 option.

50025 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
50026 results in undefined behavior.

**pthread\_attr\_getstack()**50027 **NAME**

50028 pthread\_attr\_getstack, pthread\_attr\_setstack — get and set stack attributes

50029 **SYNOPSIS**

```
50030 TSA TSS #include <pthread.h>
50031 int pthread_attr_getstack(const pthread_attr_t *restrict attr,
50032 void **restrict stackaddr, size_t *restrict stacksize);
50033 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
50034 size_t stacksize);
```

50035 **DESCRIPTION**

50036 The *pthread\_attr\_getstack()* and *pthread\_attr\_setstack()* functions, respectively, shall get and set the  
 50037 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

50038 The stack attributes specify the area of storage to be used for the created thread's stack. The base  
 50039 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be  
 50040 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD\_STACK\_MIN}. The  
 50041 *pthread\_attr\_setstack()* function may fail with [EINVAL] if *stackaddr* does not meet  
 50042 implementation-defined alignment requirements. All pages within the stack described by  
 50043 *stackaddr* and *stacksize* shall be both readable and writable by the thread.

50044 If the *pthread\_attr\_getstack()* function is called before the *stackaddr* attribute has been set, the  
 50045 behavior is unspecified.

50046 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_getstack()* or  
 50047 *pthread\_attr\_setstack()* does not refer to an initialized thread attributes object.

50048 **RETURN VALUE**

50049 Upon successful completion, these functions shall return a value of 0; otherwise, an error  
 50050 number shall be returned to indicate the error.

50051 The *pthread\_attr\_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*  
 50052 if successful.

50053 **ERRORS**

50054 The *pthread\_attr\_setstack()* function shall fail if:

50055 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds an  
 50056 implementation-defined limit.

50057 The *pthread\_attr\_setstack()* function may fail if:

50058 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or  
 50059 ((**char** \*)*stackaddr* + *stacksize*) lacks proper alignment.

50060 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable  
 50061 and writable by the thread.

50062 These functions shall not return an error code of [EINTR].

50063 **EXAMPLES**

50064 None.

50065 **APPLICATION USAGE**50066 These functions are appropriate for use by applications in an environment where the stack for a  
50067 thread must be placed in some particular region of memory.50068 While it might seem that an application could detect stack overflow by providing a protected  
50069 page outside the specified stack region, this cannot be done portably. Implementations are free  
50070 to place the thread's initial stack pointer anywhere within the specified region to accommodate  
50071 the machine's stack pointer behavior and allocation requirements. Furthermore, on some  
50072 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers  
50073 allocated within the region will overlap somewhere in the middle of the region.50074 After a successful call to `pthread_attr_setstack()`, the storage area specified by the `stackaddr`  
50075 parameter is under the control of the implementation, as described in Section 2.9.8 (on page 516).50076 The specification of the `stackaddr` attribute presents several ambiguities that make portable use of  
50077 these functions impossible. For example, the standard allows implementations to impose  
50078 arbitrary alignment requirements on `stackaddr`. Applications cannot assume that a buffer  
50079 obtained from `malloc()` is suitably aligned. Note that although the `stacksize` value passed to  
50080 `pthread_attr_setstack()` must satisfy alignment requirements, the same is not true for  
50081 `pthread_attr_setstacksize()` where the implementation must increase the specified size if necessary  
50082 to achieve the proper alignment.50083 **RATIONALE**50084 If an implementation detects that the value specified by the `attr` argument to  
50085 `pthread_attr_getstack()` or `pthread_attr_setstack()` does not refer to an initialized thread attributes  
50086 object, it is recommended that the function should fail and report an [EINVAL] error.50087 **FUTURE DIRECTIONS**

50088 None.

50089 **SEE ALSO**50090 `pthread_attr_destroy()`, `pthread_attr_getdetachstate()`, `pthread_attr_getstacksize()`, `pthread_create()`50091 XBD `<limits.h>`, `<pthread.h>`50092 **CHANGE HISTORY**50093 First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE  
50094 PASC Interpretation 1003.1 #101.50095 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the  
50096 APPLICATION USAGE section to refer to Section 2.9.8 (on page 516).50097 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS  
50098 section to include optional errors for the case when `attr` refers to an uninitialized thread attribute  
50099 object.50100 **Issue 7**50101 SD5-XSH-ERN-66 is applied, correcting the use of `attr` in the [EINVAL] error condition.50102 Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is  
50103 called before the `stackaddr` attribute is set.

50104 SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

50105 The description of the `stackaddr` attribute is updated in the DESCRIPTION and APPLICATION  
50106 USAGE sections.

## **pthread\_attr\_getstack()**

*System Interfaces*

50107  
50108

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50109 **NAME**

50110 pthread\_attr\_getstacksize, pthread\_attr\_setstacksize — get and set the stacksize attribute

50111 **SYNOPSIS**

```
50112 TSS #include <pthread.h>
50113 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
50114 size_t *restrict stacksize);
50115 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

50116 **DESCRIPTION**50117 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions, respectively, shall get and  
50118 set the thread creation *stacksize* attribute in the *attr* object.50119 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created  
50120 threads stack.50121 The behavior is undefined if the value specified by the *attr* argument to  
50122 *pthread\_attr\_getstacksize()* or *pthread\_attr\_setstacksize()* does not refer to an initialized thread  
50123 attributes object.50124 **RETURN VALUE**50125 Upon successful completion, *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* shall  
50126 return a value of 0; otherwise, an error number shall be returned to indicate the error.50127 The *pthread\_attr\_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if  
50128 successful.50129 **ERRORS**50130 The *pthread\_attr\_setstacksize()* function shall fail if:50131 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds a  
50132 system-imposed limit.

50133 These functions shall not return an error code of [EINTR].

50134 **EXAMPLES**

50135 None.

50136 **APPLICATION USAGE**

50137 None.

50138 **RATIONALE**50139 If an implementation detects that the value specified by the *attr* argument to  
50140 *pthread\_attr\_getstacksize()* or *pthread\_attr\_setstacksize()* does not refer to an initialized thread  
50141 attributes object, it is recommended that the function should fail and report an [EINVAL] error.50142 **FUTURE DIRECTIONS**

50143 None.

50144 **SEE ALSO**50145 *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*

50146 XBD &lt;limits.h&gt;, &lt;pthread.h&gt;

50147 **CHANGE HISTORY**

50148 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**pthread\_attr\_getstacksize()**

System Interfaces

50149 **Issue 6**

50150 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked as part of the  
50151 Threads and Thread Stack Size Attribute options.

50152 The **restrict** keyword is added to the *pthread\_attr\_getstacksize()* prototype for alignment with the  
50153 ISO/IEC 9899:1999 standard.

50154 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code  
50155 in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack  
50156 Address Attribute” option to “Thread Stack Size Attribute” option.

50157 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS  
50158 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
50159 object.

50160 **Issue 7**

50161 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are moved from the  
50162 Threads option.

50163 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
50164 results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread\_attr\_init()**50165 **NAME**

50166 pthread\_attr\_init — initialize the thread attributes object

50167 **SYNOPSIS**

50168 #include &lt;pthread.h&gt;

50169 int pthread\_attr\_init(pthread\_attr\_t \*attr);

50170 **DESCRIPTION**50171 Refer to *pthread\_attr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_attr\_setdetachstate()***System Interfaces*50172 **NAME**

50173 pthread\_attr\_setdetachstate — set the detachstate attribute

50174 **SYNOPSIS**

50175 #include &lt;pthread.h&gt;

50176 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

50177 **DESCRIPTION**50178 Refer to *pthread\_attr\_getdetachstate()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread\_attr\_setguardsize()**50179 **NAME**

50180 pthread\_attr\_setguardsize — set the thread guardsize attribute

50181 **SYNOPSIS**

50182 #include &lt;pthread.h&gt;

50183 int pthread\_attr\_setguardsize(pthread\_attr\_t \*attr,  
50184 size\_t guardsize);50185 **DESCRIPTION**50186 Refer to *pthread\_attr\_getguardsize()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_attr\_setinheritsched()**

System Interfaces

50187 **NAME**50188 pthread\_attr\_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)50189 **SYNOPSIS**

```
50190 TPS #include <pthread.h>
50191 int pthread_attr_setinheritsched(pthread_attr_t *attr,
50192 int inheritsched);
```

50193 **DESCRIPTION**50194 Refer to [pthread\\_attr\\_getinheritsched\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50195 **NAME**

50196 pthread\_attr\_setschedparam — set the schedparam attribute

50197 **SYNOPSIS**

50198 #include &lt;pthread.h&gt;

50199 int pthread\_attr\_setschedparam(pthread\_attr\_t \*restrict attr,  
50200 const struct sched\_param \*restrict param);50201 **DESCRIPTION**50202 Refer to [pthread\\_attr\\_getschedparam\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_attr\_setschedpolicy()**

System Interfaces

50203 **NAME**50204 pthread\_attr\_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)50205 **SYNOPSIS**

50206 TPS #include &lt;pthread.h&gt;

50207 int pthread\_attr\_setschedpolicy(pthread\_attr\_t \*attr, int policy);

50208 **DESCRIPTION**50209 Refer to *pthread\_attr\_getschedpolicy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50210 **NAME**50211 pthread\_attr\_setscope — set the contention scope attribute (**REALTIME THREADS**)50212 **SYNOPSIS**

50213 TPS #include &lt;pthread.h&gt;

50214 int pthread\_attr\_setscope(pthread\_attr\_t \*attr, int contention\_scope);

50215 **DESCRIPTION**50216 Refer to *pthread\_attr\_getscope()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_attr\_setstack()**

System Interfaces

50217 **NAME**

50218 pthread\_attr\_setstack — set the stack attribute

50219 **SYNOPSIS**

50220 TSA TSS #include &lt;pthread.h&gt;

50221 int pthread\_attr\_setstack(pthread\_attr\_t \*attr, void \*stackaddr,  
50222 size\_t stacksize);50223 **DESCRIPTION**50224 Refer to *pthread\_attr\_getstack()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50225 **NAME**

50226 pthread\_attr\_setstacksize — set the stacksize attribute

50227 **SYNOPSIS**

50228 TSS #include &lt;pthread.h&gt;

50229 int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize);

50230 **DESCRIPTION**50231 Refer to *pthread\_attr\_getstacksize()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_barrier\_destroy()**

System Interfaces

50232 **NAME**

50233 pthread\_barrier\_destroy, pthread\_barrier\_init — destroy and initialize a barrier object

50234 **SYNOPSIS**

```
50235 #include <pthread.h>
50236
50236 int pthread_barrier_destroy(pthread_barrier_t *barrier);
50237 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
50238 const pthread_barrierattr_t *restrict attr, unsigned count);
```

50239 **DESCRIPTION**

50240 The *pthread\_barrier\_destroy()* function shall destroy the barrier referenced by *barrier* and release  
 50241 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until  
 50242 the barrier is reinitialized by another call to *pthread\_barrier\_init()*. An implementation may use  
 50243 this function to set *barrier* to an invalid value. The results are undefined if  
 50244 *pthread\_barrier\_destroy()* is called when any thread is blocked on the barrier, or if this function is  
 50245 called with an uninitialized barrier.

50246 The *pthread\_barrier\_init()* function shall allocate any resources required to use the barrier  
 50247 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is  
 50248 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of  
 50249 a default barrier attributes object. The results are undefined if *pthread\_barrier\_init()* is called  
 50250 when any thread is blocked on the barrier (that is, has not returned from the  
 50251 *pthread\_barrier\_wait()* call). The results are undefined if a barrier is used without first being  
 50252 initialized. The results are undefined if *pthread\_barrier\_init()* is called specifying an already  
 50253 initialized barrier.

50254 The *count* argument specifies the number of threads that must call *pthread\_barrier\_wait()* before  
 50255 any of them successfully return from the call. The value specified by *count* must be greater than  
 50256 zero.

50257 If the *pthread\_barrier\_init()* function fails, the barrier shall not be initialized and the contents of  
 50258 *barrier* are undefined.

50259 Only the object referenced by *barrier* may be used for performing synchronization. The result of  
 50260 referring to copies of that object in calls to *pthread\_barrier\_destroy()* or *pthread\_barrier\_wait()* is  
 50261 undefined.

50262 **RETURN VALUE**

50263 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 50264 be returned to indicate the error.

50265 **ERRORS**

50266 The *pthread\_barrier\_init()* function shall fail if:

50267 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

50268 [EINVAL] The value specified by *count* is equal to zero.

50269 [ENOMEM] Insufficient memory exists to initialize the barrier.

50270 These functions shall not return an error code of [EINTR].

**50271 EXAMPLES**

50272       None.

**50273 APPLICATION USAGE**

50274       None.

**50275 RATIONALE**

50276       If an implementation detects that the value specified by the *barrier* argument to *pthread\_barrier\_destroy()* does not refer to an initialized barrier object, it is recommended that the function should fail and report an [EINVAL] error.

50279       If an implementation detects that the value specified by the *attr* argument to *pthread\_barrier\_init()* does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an [EINVAL] error.

50282       If an implementation detects that the value specified by the *barrier* argument to *pthread\_barrier\_destroy()* or *pthread\_barrier\_init()* refers to a barrier that is in use (for example, in a *pthread\_barrier\_wait()* call) by another thread, or detects that the value specified by the *barrier* argument to *pthread\_barrier\_init()* refers to an already initialized barrier object, it is recommended that the function should fail and report an [EBUSY] error.

**50287 FUTURE DIRECTIONS**

50288       None.

**50289 SEE ALSO**

50290       *pthread\_barrier\_wait()*

50291       XBD <pthread.h>

**50292 CHANGE HISTORY**

50293       First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

**50294 Issue 7**

50295       The *pthread\_barrier\_destroy()* and *pthread\_barrier\_init()* functions are moved from the Barriers option to the Base.

50297       The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

50299       The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed; this condition results in undefined behavior.

**pthread\_barrier\_wait()**

System Interfaces

50301 **NAME**

50302 pthread\_barrier\_wait — synchronize at a barrier

50303 **SYNOPSIS**

50304 #include &lt;pthread.h&gt;

50305 int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

50306 **DESCRIPTION**

50307 The *pthread\_barrier\_wait()* function shall synchronize participating threads at the barrier  
 50308 referenced by *barrier*. The calling thread shall block until the required number of threads have  
 50309 called *pthread\_barrier\_wait()* specifying the barrier.

50310 When the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier,  
 50311 the constant PTHREAD\_BARRIER\_SERIAL\_THREAD shall be returned to one unspecified  
 50312 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall  
 50313 be reset to the state it had as a result of the most recent *pthread\_barrier\_init()* function that  
 50314 referenced it.

50315 The constant PTHREAD\_BARRIER\_SERIAL\_THREAD is defined in <pthread.h> and its value  
 50316 shall be distinct from any other value returned by *pthread\_barrier\_wait()*.

50317 The results are undefined if this function is called with an uninitialized barrier.

50318 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the  
 50319 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the  
 50320 required number of threads have not arrived at the barrier during the execution of the signal  
 50321 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until  
 50322 the thread in the signal handler returns from it, it is unspecified whether other threads may  
 50323 proceed past the barrier once they have all reached it.

50324 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to  
 50325 use the same processing resources from eventually making forward progress in its execution.  
 50326 Eligibility for processing resources shall be determined by the scheduling policy.

50327 **RETURN VALUE**

50328 Upon successful completion, the *pthread\_barrier\_wait()* function shall return  
 50329 PTHREAD\_BARRIER\_SERIAL\_THREAD for a single (arbitrary) thread synchronized at the  
 50330 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to  
 50331 indicate the error.

50332 **ERRORS**

50333 This function shall not return an error code of [EINTR].

50334 **EXAMPLES**

50335 None.

50336 **APPLICATION USAGE**

50337 Applications using this function may be subject to priority inversion, as discussed in XBD  
 50338 Section 3.285 (on page 79).

50339 **RATIONALE**

50340 If an implementation detects that the value specified by the *barrier* argument to  
 50341 *pthread\_barrier\_wait()* does not refer to an initialized barrier object, it is recommended that the  
 50342 function should fail and report an [EINVAL] error.

50343 **FUTURE DIRECTIONS**

50344 None.

50345 **SEE ALSO**50346 [pthread\\_barrier\\_destroy\(\)](#)50347 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#)50348 **CHANGE HISTORY**

50349 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

50350 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.50351 **Issue 7**50352 The [pthread\\_barrier\\_wait\(\)](#) function is moved from the Barriers option to the Base.50353 The [EINVAL] error for an uninitialized barrier object is removed; this condition results in  
50354 undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_barrierattr\_destroy()**

System Interfaces

50355 **NAME**

50356 `pthread_barrierattr_destroy`, `pthread_barrierattr_init` — destroy and initialize the barrier  
 50357 attributes object

50358 **SYNOPSIS**

```
50359 #include <pthread.h>
50360 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
50361 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

50362 **DESCRIPTION**

50363 The `pthread_barrierattr_destroy()` function shall destroy a barrier attributes object. A destroyed  
 50364 `attr` attributes object can be reinitialized using `pthread_barrierattr_init()`; the results of otherwise  
 50365 referencing the object after it has been destroyed are undefined. An implementation may cause  
 50366 `pthread_barrierattr_destroy()` to set the object referenced by `attr` to an invalid value.

50367 The `pthread_barrierattr_init()` function shall initialize a barrier attributes object `attr` with the  
 50368 default value for all of the attributes defined by the implementation.

50369 If `pthread_barrierattr_init()` is called specifying an already initialized `attr` attributes object, the  
 50370 results are undefined.

50371 After a barrier attributes object has been used to initialize one or more barriers, any function  
 50372 affecting the attributes object (including destruction) shall not affect any previously initialized  
 50373 barrier.

50374 The behavior is undefined if the value specified by the `attr` argument to  
 50375 `pthread_barrierattr_destroy()` does not refer to an initialized barrier attributes object.

50376 **RETURN VALUE**

50377 If successful, the `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions shall return  
 50378 zero; otherwise, an error number shall be returned to indicate the error.

50379 **ERRORS**

50380 The `pthread_barrierattr_init()` function shall fail if:

50381 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

50382 These functions shall not return an error code of [EINTR].

50383 **EXAMPLES**

50384 None.

50385 **APPLICATION USAGE**

50386 None.

50387 **RATIONALE**

50388 If an implementation detects that the value specified by the `attr` argument to  
 50389 `pthread_barrierattr_destroy()` does not refer to an initialized barrier attributes object, it is  
 50390 recommended that the function should fail and report an [EINVAL] error.

50391 **FUTURE DIRECTIONS**

50392 None.

50393 **SEE ALSO**

50394 [\*pthread\\_barrierattr\\_getpshared\(\)\*](#)

50395 XBD [\*<pthread.h>\*](#)

50396 **CHANGE HISTORY**

50397 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

50398 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.50399 **Issue 7**50400 The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the Barriers option to the Base.

50402 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

50403

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_barrierattr\_getpshared()**

System Interfaces

50404 **NAME**

50405 pthread\_barrierattr\_getpshared, pthread\_barrierattr\_setpshared — get and set the process-  
 50406 shared attribute of the barrier attributes object

50407 **SYNOPSIS**

```
50408 TSH #include <pthread.h>
50409
50409 int pthread_barrierattr_getpshared(const pthread_barrierattr_t
50410     *restrict attr, int *restrict pshared);
50411 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
50412     int pshared);
```

50413 **DESCRIPTION**

50414 The *pthread\_barrierattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 50415 from the attributes object referenced by *attr*. The *pthread\_barrierattr\_setpshared()* function shall  
 50416 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

50417 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a barrier to be  
 50418 operated upon by any thread that has access to the memory where the barrier is allocated. If the  
 50419 *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the barrier shall only be operated  
 50420 upon by threads created within the same process as the thread that initialized the barrier; if  
 50421 threads of different processes attempt to operate on such a barrier, the behavior is undefined.  
 50422 The default value of the attribute shall be `PTHREAD_PROCESS_PRIVATE`. Both constants  
 50423 `PTHREAD_PROCESS_SHARED` and `PTHREAD_PROCESS_PRIVATE` are defined in  
 50424 `<pthread.h>`.

50425 Additional attributes, their default values, and the names of the associated functions to get and  
 50426 set those attribute values are implementation-defined.

50427 The behavior is undefined if the value specified by the *attr* argument to  
 50428 *pthread\_barrierattr\_getpshared()* or *pthread\_barrierattr\_setpshared()* does not refer to an initialized  
 50429 barrier attributes object.

50430 **RETURN VALUE**

50431 If successful, the *pthread\_barrierattr\_getpshared()* function shall return zero and store the value of  
 50432 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 50433 an error number shall be returned to indicate the error.

50434 If successful, the *pthread\_barrierattr\_setpshared()* function shall return zero; otherwise, an error  
 50435 number shall be returned to indicate the error.

50436 **ERRORS**

50437 The *pthread\_barrierattr\_setpshared()* function may fail if:

50438 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal  
 50439 values `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

50440 These functions shall not return an error code of [EINTR].

**50441 EXAMPLES**

50442       None.

**50443 APPLICATION USAGE**

50444       The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are part of the  
50445       Thread Process-Shared Synchronization option and need not be provided on all  
50446       implementations.

**50447 RATIONALE**

50448       If an implementation detects that the value specified by the *attr* argument to  
50449       *pthread\_barrierattr\_getpshared()* or *pthread\_barrierattr\_setpshared()* does not refer to an initialized  
50450       barrier attributes object, it is recommended that the function should fail and report an [EINVAL]  
50451       error.

**50452 FUTURE DIRECTIONS**

50453       None.

**50454 SEE ALSO**

50455       *pthread\_barrier\_destroy()*, *pthread\_barrierattr\_destroy()*

50456       XBD <pthread.h>

**50457 CHANGE HISTORY**

50458       First released in Issue 6. Derived from IEEE Std 1003.1j-2000

**50459 Issue 7**

50460       The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are moved from  
50461       the Barriers option.

50462       The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition  
50463       results in undefined behavior.

**pthread\_barrierattr\_init()***System Interfaces*50464 **NAME**

50465 pthread\_barrierattr\_init — initialize the barrier attributes object

50466 **SYNOPSIS**

50467 #include &lt;pthread.h&gt;

50468 int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

50469 **DESCRIPTION**50470 Refer to *pthread\_barrierattr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50471 **NAME**

50472 pthread\_barrierattr\_setpshared — set the process-shared attribute of the barrier attributes object

50473 **SYNOPSIS**

```
50474 TSH #include <pthread.h>
50475      int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
50476      int pshared);
```

50477 **DESCRIPTION**50478 Refer to [pthread\\_barrierattr\\_getpshared\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_cancel()**50479 **NAME**

50480 pthread\_cancel — cancel execution of a thread

50481 **SYNOPSIS**

50482 #include &lt;pthread.h&gt;

50483 int pthread\_cancel(pthread\_t thread);

50484 **DESCRIPTION**

50485 The *pthread\_cancel()* function shall request that *thread* be canceled. The target thread's  
 50486 cancelability state and type determines when the cancellation takes effect. When the cancellation  
 50487 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last  
 50488 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called  
 50489 for *thread*. When the last destructor function returns, *thread* shall be terminated.

50490 The cancellation processing in the target thread shall run asynchronously with respect to the  
 50491 calling thread returning from *pthread\_cancel()*.

50492 **RETURN VALUE**

50493 If successful, the *pthread\_cancel()* function shall return zero; otherwise, an error number shall be  
 50494 returned to indicate the error.

50495 **ERRORS**

50496 The *pthread\_cancel()* function shall not return an error code of [EINTR].

50497 **EXAMPLES**

50498 None.

50499 **APPLICATION USAGE**

50500 None.

50501 **RATIONALE**

50502 Two alternative functions were considered for sending the cancellation notification to a thread.  
 50503 One would be to define a new SIGCANCEL signal that had the cancellation semantics when  
 50504 delivered; the other was to define the new *pthread\_cancel()* function, which would trigger the  
 50505 cancellation semantics.

50506 The advantage of a new signal was that so much of the delivery criteria were identical to that  
 50507 used when trying to deliver a signal that making cancellation notification a signal was seen as  
 50508 consistent. Indeed, many implementations implement cancellation using a special signal. On the  
 50509 other hand, there would be no signal functions that could be used with this signal except  
 50510 *pthread\_kill()*, and the behavior of the delivered cancellation signal would be unlike any  
 50511 previously existing defined signal.

50512 The benefits of a special function include the recognition that this signal would be defined  
 50513 because of the similar delivery criteria and that this is the only common behavior between a  
 50514 cancellation request and a signal. In addition, the cancellation delivery mechanism does not  
 50515 have to be implemented as a signal. There are also strong, if not stronger, parallels with  
 50516 language exception mechanisms than with signals that are potentially obscured if the delivery  
 50517 mechanism is visibly closer to signals.

50518 In the end, it was considered that as there were so many exceptions to the use of the new signal  
 50519 with existing signals functions it would be misleading. A special function has resolved this  
 50520 problem. This function was carefully defined so that an implementation wishing to provide the  
 50521 cancellation functions on top of signals could do so. The special function also means that  
 50522 implementations are not obliged to implement cancellation with signals.

50523 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
 50524 that the function should fail and report an [ESRCH] error.

50525 **FUTURE DIRECTIONS**

50526 None.

50527 **SEE ALSO**50528 *pthread\_exit()*, *pthread\_cond\_timedwait()*, *pthread\_join()*, *pthread\_setcancelstate()*

50529 XBD &lt;pthread.h&gt;

50530 **CHANGE HISTORY**

50531 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50532 **Issue 6**50533 The *pthread\_cancel()* function is marked as part of the Threads option.50534 **Issue 7**50535 The *pthread\_cancel()* function is moved from the Threads option to the Base.

50536 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_cleanup\_pop()**50537 **NAME**

50538 pthread\_cleanup\_pop, pthread\_cleanup\_push — establish cancellation handlers

50539 **SYNOPSIS**

50540 #include &lt;pthread.h&gt;

50541 void pthread\_cleanup\_pop(int *execute*);50542 void pthread\_cleanup\_push(void (\**routine*)(void\*), void \**arg*);50543 **DESCRIPTION**50544 The *pthread\_cleanup\_pop()* function shall remove the routine at the top of the calling thread's  
50545 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).50546 The *pthread\_cleanup\_push()* function shall push the specified cancellation cleanup handler *routine*  
50547 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be  
50548 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

- 50549
- The thread exits (that is, calls *pthread\_exit()*).
50550   - The thread acts upon a cancellation request.
50551   - The thread calls *pthread\_cleanup\_pop()* with a non-zero *execute* argument.

50552 These functions may be implemented as macros. The application shall ensure that they appear  
50553 as statements, and in pairs within the same lexical scope (that is, the *pthread\_cleanup\_push()*  
50554 macro may be thought to expand to a token list whose first token is '{' with  
50555 *pthread\_cleanup\_pop()* expanding to a token list whose last token is the corresponding '}').50556 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to  
50557 *pthread\_cleanup\_push()* or *pthread\_cleanup\_pop()* made without the matching call since the jump  
50558 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation  
50559 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation  
50560 cleanup handler.50561 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block  
50562 described by a pair of *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions calls is  
50563 undefined.50564 **RETURN VALUE**50565 The *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions shall not return a value.50566 **ERRORS**

50567 No errors are defined.

50568 These functions shall not return an error code of [EINTR].

50569 **EXAMPLES**50570 The following is an example using thread primitives to implement a cancelable, writers-priority  
50571 read-write lock:50572 typedef struct {  
50573 pthread\_mutex\_t lock;  
50574 pthread\_cond\_t rcond,  
50575 wcond;  
50576 int lock\_count; /\* < 0 .. Held by writer. \*/  
50577 /\* > 0 .. Held by lock\_count readers. \*/  
50578 /\* = 0 .. Held by nobody. \*/  
50579 int waiting\_writers; /\* Count of waiting writers. \*/  
50580 } rwlock;

```

50581     void
50582     waiting_reader_cleanup(void *arg)
50583     {
50584         rwlock *l;
50585
50586         l = (rwlock *) arg;
50587         pthread_mutex_unlock(&l->lock);
50588     }
50589
50590     void
50591     lock_for_read(rwlock *l)
50592     {
50593         pthread_mutex_lock(&l->lock);
50594         pthread_cleanup_push(waiting_reader_cleanup, l);
50595         while ((l->lock_count < 0) && (l->waiting_writers != 0))
50596             pthread_cond_wait(&l->rcond, &l->lock);
50597         l->lock_count++;
50598         /*
50599          * Note the pthread_cleanup_pop executes
50600          * waiting_reader_cleanup.
50601          */
50602         pthread_cleanup_pop(1);
50603     }
50604
50605     void
50606     release_read_lock(rwlock *l)
50607     {
50608         pthread_mutex_lock(&l->lock);
50609         if (--l->lock_count == 0)
50610             pthread_cond_signal(&l->wcond);
50611         pthread_mutex_unlock(l);
50612     }
50613
50614     void
50615     waiting_writer_cleanup(void *arg)
50616     {
50617         rwlock *l;
50618
50619         l = (rwlock *) arg;
50620         if ((l->waiting_writers == 0) && (l->lock_count >= 0)) {
50621             /*
50622              * This only happens if we have been canceled.
50623              */
50624             pthread_cond_broadcast(&l->wcond);
50625         }
50626         pthread_mutex_unlock(&l->lock);
50627     }
50628
50629     void
50630     lock_for_write(rwlock *l)
50631     {
50632         pthread_mutex_lock(&l->lock);
50633         l->waiting_writers++;
50634         pthread_cleanup_push(waiting_writer_cleanup, l);
50635         while (l->lock_count != 0)

```

**pthread\_cleanup\_pop()**

```

50630         pthread_cond_wait(&l->wcond, &l->lock);
50631         l->lock_count = -1;
50632         /*
50633          * Note the pthread_cleanup_pop executes
50634          * waiting_writer_cleanup.
50635          */
50636         pthread_cleanup_pop(1);
50637     }

50638 void
50639 release_write_lock(rwlock *l)
50640 {
50641     pthread_mutex_lock(&l->lock);
50642     l->lock_count = 0;
50643     if (l->waiting_writers == 0)
50644         pthread_cond_broadcast(&l->rcond)
50645     else
50646         pthread_cond_signal(&l->wcond);
50647     pthread_mutex_unlock(&l->lock);
50648 }

50649 /*
50650  * This function is called to initialize the read/write lock.
50651  */
50652 void
50653 initialize_rwlock(rwlock *l)
50654 {
50655     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
50656     pthread_cond_init(&l->wcond, pthread_condattr_default);
50657     pthread_cond_init(&l->rcond, pthread_condattr_default);
50658     l->lock_count = 0;
50659     l->waiting_writers = 0;
50660 }

50661 reader_thread()
50662 {
50663     lock_for_read(&lock);
50664     pthread_cleanup_push(release_read_lock, &lock);
50665     /*
50666      * Thread has read lock.
50667      */
50668     pthread_cleanup_pop(1);
50669 }

50670 writer_thread()
50671 {
50672     lock_for_write(&lock);
50673     pthread_cleanup_push(release_write_lock, &lock);
50674     /*
50675      * Thread has write lock.
50676      */
50677     pthread_cleanup_pop(1);
50678 }

```

**50679 APPLICATION USAGE**

50680 The two routines that push and pop cancellation cleanup handlers, *pthread\_cleanup\_push()* and  
 50681 *pthread\_cleanup\_pop()*, can be thought of as left and right-parentheses. They always need to be  
 50682 matched.

**50683 RATIONALE**

50684 The restriction that the two routines that push and pop cancellation cleanup handlers,  
 50685 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()*, have to appear in the same lexical scope  
 50686 allows for efficient macro or compiler implementations and efficient storage management. A  
 50687 sample implementation of these routines as macros might look like this:

```
50688 #define pthread_cleanup_push(rtn, arg) { \
50689     struct _pthread_handler_rec __cleanup_handler, *__head; \
50690     __cleanup_handler.rtn = rtn; \
50691     __cleanup_handler.arg = arg; \
50692     (void) pthread_getspecific(_pthread_handler_key, &__head); \
50693     __cleanup_handler.next = *__head; \
50694     *__head = &__cleanup_handler;
50695
50696 #define pthread_cleanup_pop(ex) \
50697     *__head = __cleanup_handler.next; \
50698     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
50699 }
```

50699 A more ambitious implementation of these routines might do even better by allowing the  
 50700 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

50701 This volume of POSIX.1-2008 currently leaves unspecified the effect of calling *longjmp()* from a  
 50702 signal handler executing in a POSIX System Interfaces function. If an implementation wants to  
 50703 allow this and give the programmer reasonable behavior, the *longjmp()* function has to call all  
 50704 cancellation cleanup handlers that have been pushed but not popped since the time *setjmp()* was  
 50705 called.

50706 Consider a multi-threaded function called by a thread that uses signals. If a signal were  
 50707 delivered to a signal handler during the operation of *qsort()* and that handler were to call  
 50708 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads  
 50709 created by the *qsort()* function would not be canceled. Instead, they would continue to execute  
 50710 and write into the argument array even though the array might have been popped off the stack.

50711 Note that the specified cleanup handling mechanism is especially tied to the C language and,  
 50712 while the requirement for a uniform mechanism for expressing cleanup is language-  
 50713 independent, the mechanism used in other languages may be quite different. In addition, this  
 50714 mechanism is really only necessary due to the lack of a real exception mechanism in the C  
 50715 language, which would be the ideal solution.

50716 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation  
 50717 points in its signal handlers, blocks any signal whose handler may have cancellation points  
 50718 while calling async-unsafe functions, or disables cancellation while calling async-unsafe  
 50719 functions, all functions may be safely called from cancellation cleanup routines.

**50720 FUTURE DIRECTIONS**

50721 None.

**pthread\_cleanup\_pop()**

System Interfaces

50722 **SEE ALSO**50723 *pthread\_cancel()*, *pthread\_setcancelstate()*

50724 XBD &lt;pthread.h&gt;

50725 **CHANGE HISTORY**

50726 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50727 **Issue 6**50728 The *pthread\_cleanup\_pop()* and *pthread\_cleanup\_push()* functions are marked as part of the  
50729 Threads option.

50730 The APPLICATION USAGE section is added.

50731 The normative text is updated to avoid use of the term “must” for application requirements.

50732 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the  
50733 DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the  
50734 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions.50735 **Issue 7**50736 The *pthread\_cleanup\_pop()* and *pthread\_cleanup\_push()* functions are moved from the Threads  
50737 option to the Base.

50738 **NAME**

50739 pthread\_cond\_broadcast, pthread\_cond\_signal — broadcast or signal a condition

50740 **SYNOPSIS**

50741 #include &lt;pthread.h&gt;

50742 int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

50743 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

50744 **DESCRIPTION**

50745 These functions shall unblock threads blocked on a condition variable.

50746 The *pthread\_cond\_broadcast()* function shall unblock all threads currently blocked on the  
50747 specified condition variable *cond*.50748 The *pthread\_cond\_signal()* function shall unblock at least one of the threads that are blocked on  
50749 the specified condition variable *cond* (if any threads are blocked on *cond*).50750 If more than one thread is blocked on a condition variable, the scheduling policy shall determine  
50751 the order in which threads are unblocked. When each thread unblocked as a result of a  
50752 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* returns from its call to *pthread\_cond\_wait()* or  
50753 *pthread\_cond\_timedwait()*, the thread shall own the mutex with which it called  
50754 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*. The thread(s) that are unblocked shall contend  
50755 for the mutex according to the scheduling policy (if applicable), and as if each had called  
50756 *pthread\_mutex\_lock()*.50757 The *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* functions may be called by a thread  
50758 whether or not it currently owns the mutex that threads calling *pthread\_cond\_wait()* or  
50759 *pthread\_cond\_timedwait()* have associated with the condition variable during their waits;  
50760 however, if predictable scheduling behavior is required, then that mutex shall be locked by the  
50761 thread calling *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()*.50762 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall have no effect if there are  
50763 no threads currently blocked on *cond*.50764 The behavior is undefined if the value specified by the *cond* argument to *pthread\_cond\_broadcast()*  
50765 or *pthread\_cond\_signal()* does not refer to an initialized condition variable.50766 **RETURN VALUE**50767 If successful, the *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall return zero;  
50768 otherwise, an error number shall be returned to indicate the error.50769 **ERRORS**

50770 These functions shall not return an error code of [EINTR].

50771 **EXAMPLES**

50772 None.

50773 **APPLICATION USAGE**50774 The *pthread\_cond\_broadcast()* function is used whenever the shared-variable state has been  
50775 changed in a way that more than one thread can proceed with its task. Consider a single  
50776 producer/multiple consumer problem, where the producer can insert multiple items on a list  
50777 that is accessed one item at a time by the consumers. By calling the *pthread\_cond\_broadcast()*  
50778 function, the producer would notify all consumers that might be waiting, and thereby the  
50779 application would receive more throughput on a multi-processor. In addition,  
50780 *pthread\_cond\_broadcast()* makes it easier to implement a read-write lock. The  
50781 *pthread\_cond\_broadcast()* function is needed in order to wake up all waiting readers when a  
50782 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function  
50783 to notify all clients of an impending transaction commit.

**pthread\_cond\_broadcast()**

50784 It is not safe to use the *pthread\_cond\_signal()* function in a signal handler that is invoked  
 50785 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean  
 50786 *pthread\_cond\_wait()* that could not be efficiently eliminated.

50787 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling  
 50788 from code running in a signal handler.

**RATIONALE**

50789 If an implementation detects that the value specified by the *cond* argument to  
 50790 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* does not refer to an initialized condition  
 50791 variable, it is recommended that the function should fail and report an [EINVAL] error.  
 50792

**Multiple Awakenings by Condition Signal**

50793  
 50794 On a multi-processor, it may be impossible for an implementation of *pthread\_cond\_signal()* to  
 50795 avoid the unblocking of more than one thread blocked on a condition variable. For example,  
 50796 consider the following partial implementation of *pthread\_cond\_wait()* and *pthread\_cond\_signal()*,  
 50797 executed by two threads in the order given. One thread is trying to wait on the condition  
 50798 variable, another is concurrently executing *pthread\_cond\_signal()*, while a third thread is already  
 50799 waiting.

```

50800 pthread_cond_wait(mutex, cond):
50801     value = cond->value; /* 1 */
50802     pthread_mutex_unlock(mutex); /* 2 */
50803     pthread_mutex_lock(cond->mutex); /* 10 */
50804     if (value == cond->value) { /* 11 */
50805         me->next_cond = cond->waiter;
50806         cond->waiter = me;
50807         pthread_mutex_unlock(cond->mutex);
50808         unable_to_run(me);
50809     } else
50810         pthread_mutex_unlock(cond->mutex); /* 12 */
50811     pthread_mutex_lock(mutex); /* 13 */

50812 pthread_cond_signal(cond):
50813     pthread_mutex_lock(cond->mutex); /* 3 */
50814     cond->value++; /* 4 */
50815     if (cond->waiter) { /* 5 */
50816         sleeper = cond->waiter; /* 6 */
50817         cond->waiter = sleeper->next_cond; /* 7 */
50818         able_to_run(sleeper); /* 8 */
50819     }
50820     pthread_mutex_unlock(cond->mutex); /* 9 */
  
```

50821 The effect is that more than one thread can return from its call to *pthread\_cond\_wait()* or  
 50822 *pthread\_cond\_timedwait()* as a result of one call to *pthread\_cond\_signal()*. This effect is called  
 50823 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that  
 50824 are so awakened is finite; for example, the next thread to call *pthread\_cond\_wait()* after the  
 50825 sequence of events above blocks.

50826 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs  
 50827 only rarely is unacceptable, especially given that one has to check the predicate associated with a  
 50828 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of  
 50829 concurrency in this basic building block for all higher-level synchronization operations.

50830 An added benefit of allowing spurious wakeups is that applications are forced to code a

50831 predicate-testing-loop around the condition wait. This also makes the application tolerate  
50832 superfluous condition broadcasts or signals on the same condition variable that may be coded in  
50833 some other part of the application. The resulting applications are thus more robust. Therefore,  
50834 POSIX.1-2008 explicitly documents that spurious wakeups may occur.

**50835 FUTURE DIRECTIONS**

50836 None.

**50837 SEE ALSO**

50838 [pthread\\_cond\\_destroy\(\)](#), [pthread\\_cond\\_timedwait\(\)](#)

50839 XBD Section 4.11 (on page 110), [<pthread.h>](#)

**50840 CHANGE HISTORY**

50841 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**50842 Issue 6**

50843 The [pthread\\_cond\\_broadcast\(\)](#) and [pthread\\_cond\\_signal\(\)](#) functions are marked as part of the  
50844 Threads option.

50845 The APPLICATION USAGE section is added.

**50846 Issue 7**

50847 The [pthread\\_cond\\_broadcast\(\)](#) and [pthread\\_cond\\_signal\(\)](#) functions are moved from the Threads  
50848 option to the Base.

50849 The [EINVAL] error for an uninitialized condition variable is removed; this condition results in  
50850 undefined behavior.

**pthread\_cond\_destroy()**

System Interfaces

50851 **NAME**

50852 pthread\_cond\_destroy, pthread\_cond\_init — destroy and initialize condition variables

50853 **SYNOPSIS**

```
50854 #include <pthread.h>
50855 int pthread_cond_destroy(pthread_cond_t *cond);
50856 int pthread_cond_init(pthread_cond_t *restrict cond,
50857     const pthread_condattr_t *restrict attr);
50858 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

50859 **DESCRIPTION**

50860 The *pthread\_cond\_destroy()* function shall destroy the given condition variable specified by *cond*;  
 50861 the object becomes, in effect, uninitialized. An implementation may cause *pthread\_cond\_destroy()*  
 50862 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can  
 50863 be reinitialized using *pthread\_cond\_init()*; the results of otherwise referencing the object after it  
 50864 has been destroyed are undefined.

50865 It shall be safe to destroy an initialized condition variable upon which no threads are currently  
 50866 blocked. Attempting to destroy a condition variable upon which other threads are currently  
 50867 blocked results in undefined behavior.

50868 The *pthread\_cond\_init()* function shall initialize the condition variable referenced by *cond* with  
 50869 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be  
 50870 used; the effect is the same as passing the address of a default condition variable attributes  
 50871 object. Upon successful initialization, the state of the condition variable shall become initialized.

50872 Only *cond* itself may be used for performing synchronization. The result of referring to copies of  
 50873 *cond* in calls to *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_signal()*,  
 50874 *pthread\_cond\_broadcast()*, and *pthread\_cond\_destroy()* is undefined.

50875 Attempting to initialize an already initialized condition variable results in undefined behavior.

50876 In cases where default condition variable attributes are appropriate, the macro  
 50877 PTHREAD\_COND\_INITIALIZER can be used to initialize condition variables that are statically  
 50878 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread\_cond\_init()*  
 50879 with parameter *attr* specified as NULL, except that no error checks are performed.

50880 The behavior is undefined if the value specified by the *cond* argument to *pthread\_cond\_destroy()*  
 50881 does not refer to an initialized condition variable.

50882 The behavior is undefined if the value specified by the *attr* argument to *pthread\_cond\_init()* does  
 50883 not refer to an initialized condition variable attributes object.

50884 **RETURN VALUE**

50885 If successful, the *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions shall return zero;  
 50886 otherwise, an error number shall be returned to indicate the error.

50887 **ERRORS**

50888 The *pthread\_cond\_init()* function shall fail if:

50889 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 50890 another condition variable.

50891 [ENOMEM] Insufficient memory exists to initialize the condition variable.

50892 These functions shall not return an error code of [EINTR].

**EXAMPLES**

50894 A condition variable can be destroyed immediately after all the threads that are blocked on it are  
50895 awakened. For example, consider the following code:

```
50896 struct list {
50897     pthread_mutex_t lm;
50898     ...
50899 }

50900 struct elt {
50901     key k;
50902     int busy;
50903     pthread_cond_t notbusy;
50904     ...
50905 }

50906 /* Find a list element and reserve it. */
50907 struct elt *
50908 list_find(struct list *lp, key k)
50909 {
50910     struct elt *ep;

50911     pthread_mutex_lock(&lp->lm);
50912     while ((ep = find_elt(l, k) != NULL) && ep->busy)
50913         pthread_cond_wait(&ep->notbusy, &lp->lm);
50914     if (ep != NULL)
50915         ep->busy = 1;
50916     pthread_mutex_unlock(&lp->lm);
50917     return(ep);
50918 }

50919 delete_elt(struct list *lp, struct elt *ep)
50920 {
50921     pthread_mutex_lock(&lp->lm);
50922     assert(ep->busy);
50923     ... remove ep from list ...
50924     ep->busy = 0; /* Paranoid. */
50925     (A) pthread_cond_broadcast(&ep->notbusy);
50926     pthread_mutex_unlock(&lp->lm);
50927     (B) pthread_cond_destroy(&rp->notbusy);
50928     free(ep);
50929 }
```

50930 In this example, the condition variable and its list element may be freed (line B) immediately  
50931 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that  
50932 no other thread can touch the element to be deleted.

**APPLICATION USAGE**

50934 None.

**RATIONALE**

50936 If an implementation detects that the value specified by the *cond* argument to  
50937 *pthread\_cond\_destroy()* does not refer to an initialized condition variable, it is recommended that  
50938 the function should fail and report an [EINVAL] error.

50939 If an implementation detects that the value specified by the *cond* argument to

**pthread\_cond\_destroy()**

System Interfaces

50940 *pthread\_cond\_destroy()* or *pthread\_cond\_init()* refers to a condition variable that is in use (for  
 50941 example, in a *pthread\_cond\_wait()* call) by another thread, or detects that the value specified by  
 50942 the *cond* argument to *pthread\_cond\_init()* refers to an already initialized condition variable, it is  
 50943 recommended that the function should fail and report an [EBUSY] error.

50944 If an implementation detects that the value specified by the *attr* argument to *pthread\_cond\_init()*  
 50945 does not refer to an initialized condition variable attributes object, it is recommended that the  
 50946 function should fail and report an [EINVAL] error.

50947 See also *pthread\_mutex\_destroy()*.

**FUTURE DIRECTIONS**

50948 None.

**SEE ALSO**

50951 *pthread\_cond\_broadcast()*, *pthread\_cond\_timedwait()*, *pthread\_mutex\_destroy()*

50952 XBD <pthread.h>

**CHANGE HISTORY**

50953 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

50956 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are marked as part of the Threads  
 50957 option.

50958 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

50959 The **restrict** keyword is added to the *pthread\_cond\_init()* prototype for alignment with the  
 50960 ISO/IEC 9899:1999 standard.

**Issue 7**

50962 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are moved from the Threads option  
 50963 to the Base.

50964 The [EINVAL] error for an uninitialized condition variable and an uninitialized condition  
 50965 variable attributes object is removed; this condition results in undefined behavior.

50966 The [EBUSY] error for a condition variable already in use or an already initialized condition  
 50967 variable is removed; this condition results in undefined behavior.

*System Interfaces***pthread\_cond\_signal()**50968 **NAME**

50969 pthread\_cond\_signal — signal a condition

50970 **SYNOPSIS**

50971 #include &lt;pthread.h&gt;

50972 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

50973 **DESCRIPTION**50974 Refer to *pthread\_cond\_broadcast()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_cond\_timedwait()**50975 **NAME**

50976 pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

50977 **SYNOPSIS**

```
50978 #include <pthread.h>
50979
50979 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
50980 pthread_mutex_t *restrict mutex,
50981 const struct timespec *restrict abstime);
50982
50982 int pthread_cond_wait(pthread_cond_t *restrict cond,
50983 pthread_mutex_t *restrict mutex);
```

50984 **DESCRIPTION**

50985 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions shall block on a condition  
 50986 variable. The application shall ensure that these functions are called with *mutex* locked by the  
 50987 calling thread; otherwise, an error (for PTHREAD\_MUTEX\_ERRORCHECK and robust  
 50988 mutexes) or undefined behavior (for other mutexes) results.

50989 These functions atomically release *mutex* and cause the calling thread to block on the condition  
 50990 variable *cond*; atomically here means “atomically with respect to access by another thread to the  
 50991 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex  
 50992 after the about-to-block thread has released it, then a subsequent call to *pthread\_cond\_broadcast()*  
 50993 or *pthread\_cond\_signal()* in that thread shall behave as if it were issued after the about-to-block  
 50994 thread has blocked.

50995 Upon successful return, the mutex shall have been locked and shall be owned by the calling  
 50996 thread. If *mutex* is a robust mutex where an owner terminated while holding the lock and the  
 50997 state is recoverable, the mutex shall be acquired even though the function returns an error code.

50998 When using condition variables there is always a Boolean predicate involving shared variables  
 50999 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
 51000 from the *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* functions may occur. Since the return  
 51001 from *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* does not imply anything about the value of  
 51002 this predicate, the predicate should be re-evaluated upon such return.

51003 When a thread waits on a condition variable, having specified a particular mutex to either the  
 51004 *pthread\_cond\_timedwait()* or the *pthread\_cond\_wait()* operation, a dynamic binding is formed  
 51005 between that mutex and condition variable that remains in effect as long as at least one thread is  
 51006 blocked on the condition variable. During this time, the effect of an attempt by any thread to  
 51007 wait on that condition variable using a different mutex is undefined. Once all waiting threads  
 51008 have been unblocked (as by the *pthread\_cond\_broadcast()* operation), the next wait operation on  
 51009 that condition variable shall form a new dynamic binding with the mutex specified by that wait  
 51010 operation. Even though the dynamic binding between condition variable and mutex may be  
 51011 removed or replaced between the time a thread is unblocked from a wait on the condition  
 51012 variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked  
 51013 thread shall always re-acquire the mutex specified in the condition wait operation call from  
 51014 which it is returning.

51015 A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a  
 51016 thread is set to PTHREAD\_CANCEL\_DEFERRED, a side-effect of acting upon a cancellation  
 51017 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first  
 51018 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up  
 51019 to the point of returning from the call to *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*, but at  
 51020 that point notices the cancellation request and instead of returning to the caller of  
 51021 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*, starts the thread cancellation activities, which  
 51022 includes calling cancellation cleanup handlers.

51023 A thread that has been unblocked because it has been canceled while blocked in a call to  
 51024 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* shall not consume any condition signal that may  
 51025 be directed concurrently at the condition variable if there are other threads blocked on the  
 51026 condition variable.

51027 The *pthread\_cond\_timedwait()* function shall be equivalent to *pthread\_cond\_wait()*, except that an  
 51028 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or  
 51029 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time  
 51030 specified by *abstime* has already been passed at the time of the call.

51031 The condition variable shall have a clock attribute which specifies the clock that shall be used to  
 51032 measure the time specified by the *abstime* argument. When such timeouts occur,  
 51033 *pthread\_cond\_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*.  
 51034 The *pthread\_cond\_timedwait()* function is also a cancellation point.

51035 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal  
 51036 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it  
 51037 shall return zero due to spurious wakeup.

51038 The behavior is undefined if the value specified by the *cond* or *mutex* argument to these  
 51039 functions does not refer to an initialized condition variable or an initialized mutex object,  
 51040 respectively.

#### 51041 RETURN VALUE

51042 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed  
 51043 immediately at the beginning of processing for the function and shall cause an error return, in  
 51044 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable  
 51045 specified by *cond*.

51046 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall  
 51047 be returned to indicate the error.

#### 51048 ERRORS

51049 These functions shall fail if:

51050 [ENOTRECOVERABLE]

51051 The state protected by the mutex is not recoverable.

51052 [EOWNERDEAD]

51053 The mutex is a robust mutex and the process containing the previous owning  
 51054 thread terminated while holding the mutex lock. The mutex lock shall be  
 51055 acquired by the calling thread and it is up to the new owner to make the state  
 51056 consistent.

51057 [EPERM]

51058 The mutex type is PTHREAD\_MUTEX\_ERRORCHECK or the mutex is a  
 robust mutex, and the current thread does not own the mutex.

51059 The *pthread\_cond\_timedwait()* function shall fail if:

51060 [ETIMEDOUT] The time specified by *abstime* to *pthread\_cond\_timedwait()* has passed.

51061 [EINVAL]

51062 The *abstime* argument specified a nanosecond value less than zero or greater  
 than or equal to 1000 million.

51063 These functions may fail if:

51064 [EOWNERDEAD]

51065 The mutex is a robust mutex and the previous owning thread terminated  
 51066 while holding the mutex lock. The mutex lock shall be acquired by the calling

**pthread\_cond\_timedwait()**

51067 thread and it is up to the new owner to make the state consistent.

51068 These functions shall not return an error code of [EINTR].

**EXAMPLES**

51070 None.

**APPLICATION USAGE**

51072 Applications that have assumed that non-zero return values are errors will need updating for  
51073 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting  
51074 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error  
51075 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If  
51076 an application is supposed to work with normal and robust mutexes, it should check all return  
51077 values for error conditions and if necessary take appropriate action.

**RATIONALE**

51078 If an implementation detects that the value specified by the *cond* argument to  
51079 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* does not refer to an initialized condition  
51080 variable, or detects that the value specified by the *mutex* argument to *pthread\_cond\_timedwait()* or  
51081 *pthread\_cond\_wait()* does not refer to an initialized mutex object, it is recommended that the  
51082 function should fail and report an [EINVAL] error.  
51083

**Condition Wait Semantics**

51084  
51085 It is important to note that when *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* return  
51086 without error, the associated predicate may still be false. Similarly, when  
51087 *pthread\_cond\_timedwait()* returns with the timeout error, the associated predicate may be true  
51088 due to an unavoidable race between the expiration of the timeout and the predicate state change.

51089 The application needs to recheck the predicate on any return because it cannot be sure there is  
51090 another thread waiting on the thread to handle the signal, and if there is not then the signal is  
51091 lost. The burden is on the application to check the predicate.

51092 Some implementations, particularly on a multi-processor, may sometimes cause multiple  
51093 threads to wake up when the condition variable is signaled simultaneously on different  
51094 processors.

51095 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate  
51096 associated with the condition wait to determine whether it can safely proceed, should wait  
51097 again, or should declare a timeout. A return from the wait does not imply that the associated  
51098 predicate is either true or false.

51099 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”  
51100 that checks the predicate.

**Timed Wait Semantics**

51101  
51102 An absolute time measure was chosen for specifying the timeout parameter for two reasons.  
51103 First, a relative time measure can be easily implemented on top of a function that specifies  
51104 absolute time, but there is a race condition associated with specifying an absolute timeout on top  
51105 of a function that specifies relative timeouts. For example, assume that *clock\_gettime()* returns  
51106 the current time and *cond\_relative\_timed\_wait()* uses relative timeouts:

```
51107 clock_gettime(CLOCK_REALTIME, &now)
51108 reltime = sleep_til_this_absolute_time -now;
51109 cond_relative_timed_wait(c, m, &reltime);
```

51110 If the thread is preempted between the first statement and the last statement, the thread blocks

51111 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout  
 51112 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a  
 51113 condition wait.

51114 For cases when the system clock is advanced discontinuously by an operator, it is expected that  
 51115 implementations process any timed wait expiring at an intervening time as if that time had  
 51116 actually occurred.

#### 51117 **Cancellation and Condition Wait**

51118 A condition wait, whether timed or not, is a cancellation point. That is, the functions  
 51119 `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent)  
 51120 cancellation request is noticed. The reason for this is that an indefinite wait is possible at these  
 51121 points—whatever event is being waited for, even if the program is totally correct, might never  
 51122 occur; for example, some input data being awaited might never be sent. By making condition  
 51123 wait a cancellation point, the thread can be canceled and perform its cancellation cleanup  
 51124 handler even though it may be stuck in some indefinite wait.

51125 A side-effect of acting on a cancellation request while a thread is blocked on a condition variable  
 51126 is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in  
 51127 order to ensure that the cancellation cleanup handler is executed in the same state as the critical  
 51128 code that lies both before and after the call to the condition wait function. This rule is also  
 51129 required when interfacing to POSIX threads from languages, such as Ada or C++, which may  
 51130 choose to map cancellation onto a language exception; this rule ensures that each exception  
 51131 handler guarding a critical section can always safely depend upon the fact that the associated  
 51132 mutex has already been locked regardless of exactly where within the critical section the  
 51133 exception was raised. Without this rule, there would not be a uniform rule that exception  
 51134 handlers could follow regarding the lock, and so coding would become very cumbersome.

51135 Therefore, since *some* statement has to be made regarding the state of the lock when a  
 51136 cancellation is delivered during a wait, a definition has been chosen that makes application  
 51137 coding most convenient and error free.

51138 When acting on a cancellation request while a thread is blocked on a condition variable, the  
 51139 implementation is required to ensure that the thread does not consume any condition signals  
 51140 directed at that condition variable if there are any other threads waiting on that condition  
 51141 variable. This rule is specified in order to avoid deadlock conditions that could occur if these  
 51142 two independent requests (one acting on a thread and the other acting on the condition variable)  
 51143 were not processed independently.

#### 51144 **Performance of Mutexes and Condition Variables**

51145 Mutexes are expected to be locked only for a few instructions. This practice is almost  
 51146 automatically enforced by the desire of programmers to avoid long serial regions of execution  
 51147 (which would reduce total effective parallelism).

51148 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the  
 51149 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a  
 51150 relatively rare situation. For example, when implementing a read-write lock, code that acquires a  
 51151 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and  
 51152 return. The calling thread would actually wait on the condition variable only when there is  
 51153 already an active writer. So the efficiency of a synchronization operation is bounded by the cost  
 51154 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context  
 51155 switch.

51156 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be

**pthread\_cond\_timedwait()**

51157 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable  
 51158 is important. The cost of waiting on a condition variable should be little more than the minimal  
 51159 cost for a context switch plus the time to unlock and lock the mutex.

**51160 Features of Mutexes and Condition Variables**

51161 It had been suggested that the mutex acquisition and release be decoupled from condition wait.  
 51162 This was rejected because it is the combined nature of the operation that, in fact, facilitates  
 51163 realtime implementations. Those implementations can atomically move a high-priority thread  
 51164 between the condition variable and the mutex in a manner that is transparent to the caller. This  
 51165 can prevent extra context switches and provide more deterministic acquisition of a mutex when  
 51166 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the  
 51167 scheduling discipline. Furthermore, the current condition wait operation matches existing  
 51168 practice.

**51169 Scheduling Behavior of Mutexes and Condition Variables**

51170 Synchronization primitives that attempt to interfere with scheduling policy by specifying an  
 51171 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables  
 51172 are selected to proceed in an order dependent upon the scheduling policy rather than in some  
 51173 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which  
 51174 thread(s) are awakened and allowed to proceed.

**51175 Timed Condition Wait**

51176 The *pthread\_cond\_timedwait()* function allows an application to give up waiting for a particular  
 51177 condition after a given amount of time. An example of its use follows:

```
51178 (void) pthread_mutex_lock(&t.mn);
51179     t.waiters++;
51180     clock_gettime(CLOCK_REALTIME, &ts);
51181     ts.tv_sec += 5;
51182     rc = 0;
51183     while (! mypredicate(&t) && rc == 0)
51184         rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
51185     t.waiters--;
51186     if (rc == 0) setmystate(&t);
51187 (void) pthread_mutex_unlock(&t.mn);
```

51188 By making the timeout parameter absolute, it does not need to be recomputed each time the  
 51189 program checks its blocking predicate. If the timeout was relative, it would have to be  
 51190 recomputed before each call. This would be especially difficult since such code would need to  
 51191 take into account the possibility of extra wakeups that result from extra broadcasts or signals on  
 51192 the condition variable that occur before either the predicate is true or the timeout is due.

**51193 FUTURE DIRECTIONS**

51194 None.

**51195 SEE ALSO**

51196 [pthread\\_cond\\_broadcast\(\)](#)

51197 XBD [Section 4.11](#) (on page 110), [<pthread.h>](#)

**CHANGE HISTORY**

51198 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

51201 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are marked as part of the  
51202 Threads option.

51203 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
51204 *pthread\_cond\_wait()* function.

51205 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
51206 for the Clock Selection option.

51207 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC  
51208 Interpretation 1003.1c #28.

51209 The **restrict** keyword is added to the *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()*  
51210 prototypes for alignment with the ISO/IEC 9899:1999 standard.

51211 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the  
51212 DESCRIPTION for consistency with the *pthread\_cond\_destroy()* function that states it is safe to  
51213 destroy an initialized condition variable upon which no threads are currently blocked.

51214 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the  
51215 DESCRIPTION from “the cancelability enable state” to “the cancelability type”.

51216 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS  
51217 section to remove the error case related to *abstime* from the *pthread\_cond\_wait()* function, and to  
51218 make the error case related to *abstime* mandatory for *pthread\_cond\_timedwait()* for consistency  
51219 with other functions.

51220 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to  
51221 the RATIONALE section stating that an application should check the predicate on any return  
51222 from this function.

**Issue 7**

51223 SD5-XSH-ERN-44 is applied, changing the definition of the “shall fail” case of the [EINVAL]  
51224 error.

51226 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

51227 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are moved from the Threads  
51228 option to the Base.

51229 The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is  
51230 removed; this condition results in undefined behavior”

51231 The [EPERM] error is revised and moved to the “shall fail” list of error conditions for the  
51232 *pthread\_cond\_timedwait()* function.

51233 The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.

51234 The ERRORS section is updated to include “shall fail” cases for  
51235 PTHREAD\_MUTEX\_ERRORCHECK mutexes.

51236 The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes  
51237 where the [EPERM] error is not mandated.

**pthread\_condattr\_destroy()**51238 **NAME**

51239 pthread\_condattr\_destroy, pthread\_condattr\_init — destroy and initialize the condition variable  
51240 attributes object

51241 **SYNOPSIS**

```
51242 #include <pthread.h>
51243 int pthread_condattr_destroy(pthread_condattr_t *attr);
51244 int pthread_condattr_init(pthread_condattr_t *attr);
```

51245 **DESCRIPTION**

51246 The *pthread\_condattr\_destroy()* function shall destroy a condition variable attributes object; the  
51247 object becomes, in effect, uninitialized. An implementation may cause *pthread\_condattr\_destroy()*  
51248 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
51249 reinitialized using *pthread\_condattr\_init()*; the results of otherwise referencing the object after it  
51250 has been destroyed are undefined.

51251 The *pthread\_condattr\_init()* function shall initialize a condition variable attributes object *attr* with  
51252 the default value for all of the attributes defined by the implementation.

51253 Results are undefined if *pthread\_condattr\_init()* is called specifying an already initialized *attr*  
51254 attributes object.

51255 After a condition variable attributes object has been used to initialize one or more condition  
51256 variables, any function affecting the attributes object (including destruction) shall not affect any  
51257 previously initialized condition variables.

51258 This volume of POSIX.1-2008 requires two attributes, the *clock* attribute and the *process-shared*  
51259 attribute.

51260 Additional attributes, their default values, and the names of the associated functions to get and  
51261 set those attribute values are implementation-defined.

51262 The behavior is undefined if the value specified by the *attr* argument to  
51263 *pthread\_condattr\_destroy()* does not refer to an initialized condition variable attributes object.

51264 **RETURN VALUE**

51265 If successful, the *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions shall return  
51266 zero; otherwise, an error number shall be returned to indicate the error.

51267 **ERRORS**

51268 The *pthread\_condattr\_init()* function shall fail if:

51269 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

51270 These functions shall not return an error code of [EINTR].

51271 **EXAMPLES**

51272 None.

51273 **APPLICATION USAGE**

51274 None.

51275 **RATIONALE**

51276 A *process-shared* attribute has been defined for condition variables for the same reason it has been  
51277 defined for mutexes.

51278 If an implementation detects that the value specified by the *attr* argument to  
51279 *pthread\_condattr\_destroy()* does not refer to an initialized condition variable attributes object, it is  
51280 recommended that the function should fail and report an [EINVAL] error.

51281 See also *pthread\_attr\_destroy()* and *pthread\_mutex\_destroy()*.

51282 **FUTURE DIRECTIONS**

51283 None.

51284 **SEE ALSO**

51285 *pthread\_attr\_destroy()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
51286 *pthread\_mutex\_destroy()*

51287 XBD <pthread.h>

51288 **CHANGE HISTORY**

51289 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51290 **Issue 6**

51291 The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are marked as part of the  
51292 Threads option.

51293 **Issue 7**

51294 The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are moved from the Threads  
51295 option to the Base.

51296 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
51297 condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_condattr\_getclock()**

System Interfaces

51298 **NAME**

51299 pthread\_condattr\_getclock, pthread\_condattr\_setclock — get and set the clock selection  
 51300 condition variable attribute

51301 **SYNOPSIS**

```
51302 #include <pthread.h>
51303 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
51304 clockid_t *restrict clock_id);
51305 int pthread_condattr_setclock(pthread_condattr_t *attr,
51306 clockid_t clock_id);
```

51307 **DESCRIPTION**

51308 The *pthread\_condattr\_getclock()* function shall obtain the value of the *clock* attribute from the  
 51309 attributes object referenced by *attr*.

51310 The *pthread\_condattr\_setclock()* function shall set the *clock* attribute in an initialized attributes  
 51311 object referenced by *attr*. If *pthread\_condattr\_setclock()* is called with a *clock\_id* argument that  
 51312 refers to a CPU-time clock, the call shall fail.

51313 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of  
 51314 *pthread\_cond\_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

51315 The behavior is undefined if the value specified by the *attr* argument to  
 51316 *pthread\_condattr\_getclock()* or *pthread\_condattr\_setclock()* does not refer to an initialized condition  
 51317 variable attributes object.

51318 **RETURN VALUE**

51319 If successful, the *pthread\_condattr\_getclock()* function shall return zero and store the value of the  
 51320 clock attribute of *attr* into the object referenced by the *clock\_id* argument. Otherwise, an error  
 51321 number shall be returned to indicate the error.

51322 If successful, the *pthread\_condattr\_setclock()* function shall return zero; otherwise, an error  
 51323 number shall be returned to indicate the error.

51324 **ERRORS**

51325 The *pthread\_condattr\_setclock()* function may fail if:

51326 [EINVAL] The value specified by *clock\_id* does not refer to a known clock, or is a CPU-  
 51327 time clock.

51328 These functions shall not return an error code of [EINTR].

51329 **EXAMPLES**

51330 None.

51331 **APPLICATION USAGE**

51332 None.

51333 **RATIONALE**

51334 If an implementation detects that the value specified by the *attr* argument to  
 51335 *pthread\_condattr\_getclock()* or *pthread\_condattr\_setclock()* does not refer to an initialized condition  
 51336 variable attributes object, it is recommended that the function should fail and report an  
 51337 [EINVAL] error.

51338 **FUTURE DIRECTIONS**

51339 None.

51340 **SEE ALSO**

51341 *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*, *pthread\_condattr\_destroy()*,  
51342 *pthread\_condattr\_getpshared()*, *pthread\_create()*, *pthread\_mutex\_destroy()*

51343 XBD <pthread.h>

51344 **CHANGE HISTORY**

51345 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

51346 **Issue 7**

51347 The *pthread\_condattr\_getclock()* and *pthread\_condattr\_setclock()* functions are moved from the  
51348 Clock Selection option to the Base.

51349 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
51350 condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_condattr\_getpshared()**51351 **NAME**

51352 pthread\_condattr\_getpshared, pthread\_condattr\_setpshared — get and set the process-shared  
 51353 condition variable attributes

51354 **SYNOPSIS**

```
51355 TSH #include <pthread.h>
51356
51356 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
51357 int *restrict pshared);
51358
51358 int pthread_condattr_setpshared(pthread_condattr_t *attr,
51359 int pshared);
```

51360 **DESCRIPTION**

51361 The *pthread\_condattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 51362 from the attributes object referenced by *attr*.

51363 The *pthread\_condattr\_setpshared()* function shall set the *process-shared* attribute in an initialized  
 51364 attributes object referenced by *attr*.

51365 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a condition  
 51366 variable to be operated upon by any thread that has access to the memory where the condition  
 51367 variable is allocated, even if the condition variable is allocated in memory that is shared by  
 51368 multiple processes. If the *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the  
 51369 condition variable shall only be operated upon by threads created within the same process as the  
 51370 thread that initialized the condition variable; if threads of differing processes attempt to operate  
 51371 on such a condition variable, the behavior is undefined. The default value of the attribute is  
 51372 PTHREAD\_PROCESS\_PRIVATE.

51373 The behavior is undefined if the value specified by the *attr* argument to  
 51374 *pthread\_condattr\_getpshared()* or *pthread\_condattr\_setpshared()* does not refer to an initialized  
 51375 condition variable attributes object.

51376 **RETURN VALUE**

51377 If successful, the *pthread\_condattr\_setpshared()* function shall return zero; otherwise, an error  
 51378 number shall be returned to indicate the error.

51379 If successful, the *pthread\_condattr\_getpshared()* function shall return zero and store the value of  
 51380 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 51381 an error number shall be returned to indicate the error.

51382 **ERRORS**

51383 The *pthread\_condattr\_setpshared()* function may fail if:

51384 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 51385 for that attribute.

51386 These functions shall not return an error code of [EINTR].

51387 **EXAMPLES**

51388 None.

51389 **APPLICATION USAGE**

51390 None.

51391 **RATIONALE**

51392 If an implementation detects that the value specified by the *attr* argument to  
 51393 *pthread\_condattr\_getpshared()* or *pthread\_condattr\_setpshared()* does not refer to an initialized  
 51394 condition variable attributes object, it is recommended that the function should fail and report  
 51395 an [EINVAL] error.

51396 **FUTURE DIRECTIONS**

51397 None.

51398 **SEE ALSO**51399 *pthread\_create()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*, *pthread\_mutex\_destroy()*

51400 XBD &lt;pthread.h&gt;

51401 **CHANGE HISTORY**

51402 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51403 **Issue 6**

51404 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked as part  
 51405 of the Threads and Thread Process-Shared Synchronization options.

51406 The **restrict** keyword is added to the *pthread\_condattr\_getpshared()* prototype for alignment with  
 51407 the ISO/IEC 9899:1999 standard.

51408 **Issue 7**

51409 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are moved from the  
 51410 Threads option.

51411 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
 51412 condition results in undefined behavior.

**pthread\_condattr\_init()***System Interfaces*51413 **NAME**

51414 pthread\_condattr\_init — initialize the condition variable attributes object

51415 **SYNOPSIS**

51416 #include &lt;pthread.h&gt;

51417 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

51418 **DESCRIPTION**51419 Refer to *pthread\_condattr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51420 **NAME**

51421 pthread\_condattr\_setclock — set the clock selection condition variable attribute

51422 **SYNOPSIS**

51423 #include &lt;pthread.h&gt;

51424 int pthread\_condattr\_setclock(pthread\_condattr\_t \*attr,  
51425 clockid\_t clock\_id);51426 **DESCRIPTION**51427 Refer to [pthread\\_condattr\\_getclock\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_condattr\_setpshared()**

System Interfaces

51428 **NAME**

51429 pthread\_condattr\_setpshared — set the process-shared condition variable attribute

51430 **SYNOPSIS**

```
51431 TSH #include <pthread.h>
51432      int pthread_condattr_setpshared(pthread_condattr_t *attr,
51433      int pshared);
```

51434 **DESCRIPTION**51435 Refer to [pthread\\_condattr\\_getpshared\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51436 **NAME**

51437 pthread\_create — thread creation

51438 **SYNOPSIS**

```
51439 #include <pthread.h>
51440 int pthread_create(pthread_t *restrict thread,
51441                  const pthread_attr_t *restrict attr,
51442                  void *(*start_routine)(void*), void *restrict arg);
```

51443 **DESCRIPTION**

51444 The *pthread\_create()* function shall create a new thread, with attributes specified by *attr*, within a  
 51445 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are  
 51446 modified later, the thread's attributes shall not be affected. Upon successful completion,  
 51447 *pthread\_create()* shall store the ID of the created thread in the location referenced by *thread*.

51448 The thread is created executing *start\_routine* with *arg* as its sole argument. If the *start\_routine*  
 51449 returns, the effect shall be as if there was an implicit call to *pthread\_exit()* using the return value  
 51450 of *start\_routine* as the exit status. Note that the thread in which *main()* was originally invoked  
 51451 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call to  
 51452 *exit()* using the return value of *main()* as the exit status.

51453 The signal state of the new thread shall be initialized as follows:

- 51454 • The signal mask shall be inherited from the creating thread.
- 51455 • The set of signals pending for the new thread shall be empty.

51456 XSI The alternate stack shall not be inherited.

51457 The floating-point environment shall be inherited from the creating thread.

51458 If *pthread\_create()* fails, no new thread is created and the contents of the location referenced by  
 51459 *thread* are undefined.

51460 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock  
 51461 accessible, and the initial value of this clock shall be set to zero.

51462 The behavior is undefined if the value specified by the *attr* argument to *pthread\_create()* does not  
 51463 refer to an initialized thread attributes object.

51464 **RETURN VALUE**

51465 If successful, the *pthread\_create()* function shall return zero; otherwise, an error number shall be  
 51466 returned to indicate the error.

51467 **ERRORS**

51468 The *pthread\_create()* function shall fail if:

51469 [EAGAIN] The system lacked the necessary resources to create another thread, or the  
 51470 system-imposed limit on the total number of threads in a process  
 51471 {PTHREAD\_THREADS\_MAX} would be exceeded.

51472 [EPERM] The caller does not have appropriate privileges to set the required scheduling  
 51473 parameters or scheduling policy.

51474 The *pthread\_create()* function shall not return an error code of [EINTR].

**pthread\_create()**51475 **EXAMPLES**

51476 None.

51477 **APPLICATION USAGE**

51478 There is no requirement on the implementation that the ID of the created thread be available  
 51479 before the newly created thread starts executing. The calling thread can obtain the ID of the  
 51480 created thread through the return value of the *pthread\_create()* function, and the newly created  
 51481 thread can obtain its ID by a call to *pthread\_self()*.

51482 **RATIONALE**

51483 A suggested alternative to *pthread\_create()* would be to define two separate operations: create  
 51484 and start. Some applications would find such behavior more natural. Ada, in particular,  
 51485 separates the “creation” of a task from its “activation”.

51486 Splitting the operation was rejected by the standard developers for many reasons:

- 51487 • The number of calls required to start a thread would increase from one to two and thus
- 51488 place an additional burden on applications that do not require the additional
- 51489 synchronization. The second call, however, could be avoided by the additional
- 51490 complication of a start-up state attribute.
- 51491 • An extra state would be introduced: “created but not started”. This would require the
- 51492 standard to specify the behavior of the thread operations when the target has not yet
- 51493 started executing.
- 51494 • For those applications that require such behavior, it is possible to simulate the two separate
- 51495 steps with the facilities that are currently provided. The *start\_routine()* can synchronize by
- 51496 waiting on a condition variable that is signaled by the start operation.

51497 An Ada implementor can choose to create the thread at either of two points in the Ada program:  
 51498 when the task object is created, or when the task is activated (generally at a “begin”). If the first  
 51499 approach is adopted, the *start\_routine()* needs to wait on a condition variable to receive the order  
 51500 to begin “activation”. The second approach requires no such condition variable or extra  
 51501 synchronization. In either approach, a separate Ada task control block would need to be created  
 51502 when the task object is created to hold rendezvous queues, and so on.

51503 An extension of the preceding model would be to allow the state of the thread to be modified  
 51504 between the create and start. This would allow the thread attributes object to be eliminated. This  
 51505 has been rejected because:

- 51506 • All state in the thread attributes object has to be able to be set for the thread. This would
- 51507 require the definition of functions to modify thread attributes. There would be no
- 51508 reduction in the number of function calls required to set up the thread. In fact, for an
- 51509 application that creates all threads using identical attributes, the number of function calls
- 51510 required to set up the threads would be dramatically increased. Use of a thread attributes
- 51511 object permits the application to make one set of attribute setting function calls.
- 51512 Otherwise, the set of attribute setting function calls needs to be made for each thread
- 51513 creation.
- 51514 • Depending on the implementation architecture, functions to set thread state would require
- 51515 kernel calls, or for other implementation reasons would not be able to be implemented as
- 51516 macros, thereby increasing the cost of thread creation.
- 51517 • The ability for applications to segregate threads by class would be lost.

51518 Another suggested alternative uses a model similar to that for process creation, such as “thread  
 51519 fork”. The fork semantics would provide more flexibility and the “create” function can be  
 51520 implemented simply by doing a thread fork followed immediately by a call to the desired “start

51521 routine” for the thread. This alternative has these problems:

- 51522 • For many implementations, the entire stack of the calling thread would need to be
- 51523 duplicated, since in many architectures there is no way to determine the size of the calling
- 51524 frame.
- 51525 • Efficiency is reduced since at least some part of the stack has to be copied, even though in
- 51526 most cases the thread never needs the copied context, since it merely calls the desired start
- 51527 routine.

51528 If an implementation detects that the value specified by the *attr* argument to *pthread\_create()*

51529 does not refer to an initialized thread attributes object, it is recommended that the function

51530 should fail and report an [EINVAL] error.

#### 51531 FUTURE DIRECTIONS

51532 None.

#### 51533 SEE ALSO

51534 *fork()*, *pthread\_exit()*, *pthread\_join()*

51535 XBD Section 4.11 (on page 110), <*pthread.h*>

#### 51536 CHANGE HISTORY

51537 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 51538 Issue 6

51539 The *pthread\_create()* function is marked as part of the Threads option.

51540 The following new requirements on POSIX implementations derive from alignment with the

51541 Single UNIX Specification:

- 51542 • The [EPERM] mandatory error condition is added.

51543 The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

51544 The **restrict** keyword is added to the *pthread\_create()* prototype for alignment with the

51545 ISO/IEC 9899: 1999 standard.

51546 The DESCRIPTION is updated to make it explicit that the floating-point environment is

51547 inherited from the creating thread.

51548 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the

51549 alternate stack is not inherited.

51550 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS

51551 section to remove the mandatory [EINVAL] error (“The value specified by *attr* is invalid”), and

51552 adding the optional [EINVAL] error (“The attributes specified by *attr* are invalid”).

51553 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION

51554 USAGE section.

#### 51555 Issue 7

51556 The *pthread\_create()* function is moved from the Threads option to the Base.

51557 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition

51558 results in undefined behavior.

**pthread\_detach()**51559 **NAME**

51560 pthread\_detach — detach a thread

51561 **SYNOPSIS**

51562 #include &lt;pthread.h&gt;

51563 int pthread\_detach(pthread\_t thread);

51564 **DESCRIPTION**

51565 The *pthread\_detach()* function shall indicate to the implementation that storage for the thread  
 51566 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,  
 51567 *pthread\_detach()* shall not cause it to terminate.

51568 The behavior is undefined if the value specified by the *thread* argument to *pthread\_detach()* does  
 51569 not refer to a joinable thread.

51570 **RETURN VALUE**

51571 If the call succeeds, *pthread\_detach()* shall return 0; otherwise, an error number shall be returned  
 51572 to indicate the error.

51573 **ERRORS**51574 The *pthread\_detach()* function shall not return an error code of [EINTR].51575 **EXAMPLES**

51576 None.

51577 **APPLICATION USAGE**

51578 None.

51579 **RATIONALE**

51580 The *pthread\_join()* or *pthread\_detach()* functions should eventually be called for every thread that  
 51581 is created so that storage associated with the thread may be reclaimed.

51582 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation  
 51583 attribute is sufficient, since a thread need never be dynamically detached. However, need arises  
 51584 in at least two cases:

51585 1. In a cancellation handler for a *pthread\_join()* it is nearly essential to have a  
 51586 *pthread\_detach()* function in order to detach the thread on which *pthread\_join()* was  
 51587 waiting. Without it, it would be necessary to have the handler do another *pthread\_join()* to  
 51588 attempt to detach the thread, which would both delay the cancellation processing for an  
 51589 unbounded period and introduce a new call to *pthread\_join()*, which might itself need a  
 51590 cancellation handler. A dynamic detach is nearly essential in this case.

51591 2. In order to detach the “initial thread” (as may be desirable in processes that set up server  
 51592 threads).

51593 If an implementation detects that the value specified by the *thread* argument to *pthread\_detach()*  
 51594 does not refer to a joinable thread, it is recommended that the function should fail and report an  
 51595 [EINVAL] error.

51596 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
 51597 that the function should fail and report an [ESRCH] error.

51598 **FUTURE DIRECTIONS**

51599 None.

51600 **SEE ALSO**51601 [pthread\\_join\(\)](#)51602 XBD [<pthread.h>](#)51603 **CHANGE HISTORY**

51604 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51605 **Issue 6**51606 The *pthread\_detach()* function is marked as part of the Threads option.51607 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS  
51608 section so that the [EINVAL] and [ESRCH] error cases become optional.51609 **Issue 7**51610 The *pthread\_detach()* function is moved from the Threads option to the Base.

51611 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

51612 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined  
51613 behavior.

## pthread\_equal()

**51614 NAME**

51615 pthread\_equal — compare thread IDs

**51616 SYNOPSIS**

51617 #include <pthread.h>

51618 int pthread\_equal(pthread\_t t1, pthread\_t t2);

**51619 DESCRIPTION**

51620 This function shall compare the thread IDs *t1* and *t2*.

**51621 RETURN VALUE**

51622 The *pthread\_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero shall be returned.

51624 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.

**51625 ERRORS**

51626 No errors are defined.

51627 The *pthread\_equal()* function shall not return an error code of [EINTR].

**51628 EXAMPLES**

51629 None.

**51630 APPLICATION USAGE**

51631 None.

**51632 RATIONALE**

51633 Implementations may choose to define a thread ID as a structure. This allows additional flexibility and robustness over using an **int**. For example, a thread ID could include a sequence number that allows detection of “dangling IDs” (copies of a thread ID that has been detached). Since the C language does not support comparison on structure types, the *pthread\_equal()* function is provided to compare thread IDs.

**51638 FUTURE DIRECTIONS**

51639 None.

**51640 SEE ALSO**

51641 *pthread\_create()*, *pthread\_self()*

51642 XBD <pthread.h>

**51643 CHANGE HISTORY**

51644 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**51645 Issue 6**

51646 The *pthread\_equal()* function is marked as part of the Threads option.

**51647 Issue 7**

51648 The *pthread\_equal()* function is moved from the Threads option to the Base.

**51649 NAME**

51650 pthread\_exit — thread termination

**51651 SYNOPSIS**

51652 #include &lt;pthread.h&gt;

51653 void pthread\_exit(void \*value\_ptr);

**51654 DESCRIPTION**

51655 The *pthread\_exit()* function shall terminate the calling thread and make the value *value\_ptr*  
 51656 available to any successful join with the terminating thread. Any cancellation cleanup handlers  
 51657 that have been pushed and not yet popped shall be popped in the reverse order that they were  
 51658 pushed and then executed. After all cancellation cleanup handlers have been executed, if the  
 51659 thread has any thread-specific data, appropriate destructor functions shall be called in an  
 51660 unspecified order. Thread termination does not release any application visible process resources,  
 51661 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level  
 51662 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

51663 An implicit call to *pthread\_exit()* is made when a thread other than the thread in which *main()*  
 51664 was first invoked returns from the start routine that was used to create it. The function's return  
 51665 value shall serve as the thread's exit status.

51666 The behavior of *pthread\_exit()* is undefined if called from a cancellation cleanup handler or  
 51667 destructor function that was invoked as a result of either an implicit or explicit call to  
 51668 *pthread\_exit()*.

51669 After a thread has terminated, the result of access to local (auto) variables of the thread is  
 51670 undefined. Thus, references to local variables of the exiting thread should not be used for the  
 51671 *pthread\_exit()* *value\_ptr* parameter value.

51672 The process shall exit with an exit status of 0 after the last thread has been terminated. The  
 51673 behavior shall be as if the implementation called *exit()* with a zero argument at thread  
 51674 termination time.

**51675 RETURN VALUE**51676 The *pthread\_exit()* function cannot return to its caller.**51677 ERRORS**

51678 No errors are defined.

**51679 EXAMPLES**

51680 None.

**51681 APPLICATION USAGE**

51682 None.

**51683 RATIONALE**

51684 The normal mechanism by which a thread terminates is to return from the routine that was  
 51685 specified in the *pthread\_create()* call that started it. The *pthread\_exit()* function provides the  
 51686 capability for a thread to terminate without requiring a return from the start routine of that  
 51687 thread, thereby providing a function analogous to *exit()*.

51688 Regardless of the method of thread termination, any cancellation cleanup handlers that have  
 51689 been pushed and not yet popped are executed, and the destructors for any existing thread-  
 51690 specific data are executed. This volume of POSIX.1-2008 requires that cancellation cleanup  
 51691 handlers be popped and called in order. After all cancellation cleanup handlers have been  
 51692 executed, thread-specific data destructors are called, in an unspecified order, for each item of  
 51693 thread-specific data that exists in the thread. This ordering is necessary because cancellation  
 51694 cleanup handlers may rely on thread-specific data.

**pthread\_exit()**

51695 As the meaning of the status is determined by the application (except when the thread has been  
51696 canceled, in which case it is PTHREAD\_CANCELED), the implementation has no idea what an  
51697 illegal status value is, which is why no address error checking is done.

**51698 FUTURE DIRECTIONS**

51699 None.

**51700 SEE ALSO**

51701 *exit()*, *pthread\_create()*, *pthread\_join()*

51702 XBD <pthread.h>

**51703 CHANGE HISTORY**

51704 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**51705 Issue 6**

51706 The *pthread\_exit()* function is marked as part of the Threads option.

**51707 Issue 7**

51708 The *pthread\_exit()* function is moved from the Threads option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51709 **NAME**

51710 pthread\_getconcurrency, pthread\_setconcurrency — get and set the level of concurrency

51711 **SYNOPSIS**

```
51712 OB XSI #include <pthread.h>
51713 int pthread_getconcurrency(void);
51714 int pthread_setconcurrency(int new_level);
```

51715 **DESCRIPTION**

51716 Unbound threads in a process may or may not be required to be simultaneously active. By  
 51717 default, the threads implementation ensures that a sufficient number of threads are active so that  
 51718 the process can continue to make progress. While this conserves system resources, it may not  
 51719 produce the most effective level of concurrency.

51720 The *pthread\_setconcurrency()* function allows an application to inform the threads  
 51721 implementation of its desired concurrency level, *new\_level*. The actual level of concurrency  
 51722 provided by the implementation as a result of this function call is unspecified.

51723 If *new\_level* is zero, it causes the implementation to maintain the concurrency level at its  
 51724 discretion as if *pthread\_setconcurrency()* had never been called.

51725 The *pthread\_getconcurrency()* function shall return the value set by a previous call to the  
 51726 *pthread\_setconcurrency()* function. If the *pthread\_setconcurrency()* function was not previously  
 51727 called, this function shall return zero to indicate that the implementation is maintaining the  
 51728 concurrency level.

51729 A call to *pthread\_setconcurrency()* shall inform the implementation of its desired concurrency  
 51730 level. The implementation shall use this as a hint, not a requirement.

51731 If an implementation does not support multiplexing of user threads on top of several kernel-  
 51732 scheduled entities, the *pthread\_setconcurrency()* and *pthread\_getconcurrency()* functions are  
 51733 provided for source code compatibility but they shall have no effect when called. To maintain  
 51734 the function semantics, the *new\_level* parameter is saved when *pthread\_setconcurrency()* is called  
 51735 so that a subsequent call to *pthread\_getconcurrency()* shall return the same value.

51736 **RETURN VALUE**

51737 If successful, the *pthread\_setconcurrency()* function shall return zero; otherwise, an error number  
 51738 shall be returned to indicate the error.

51739 The *pthread\_getconcurrency()* function shall always return the concurrency level set by a previous  
 51740 call to *pthread\_setconcurrency()*. If the *pthread\_setconcurrency()* function has never been called,  
 51741 *pthread\_getconcurrency()* shall return zero.

51742 **ERRORS**

51743 The *pthread\_setconcurrency()* function shall fail if:

51744 [EINVAL] The value specified by *new\_level* is negative.

51745 [EAGAIN] The value specified by *new\_level* would cause a system resource to be  
 51746 exceeded.

51747 The *pthread\_setconcurrency()* function shall not return an error code of [EINTR].

**pthread\_getconcurrency()**51748 **EXAMPLES**

51749 None.

51750 **APPLICATION USAGE**

51751 Application developers should note that an implementation can always ignore any calls to  
51752 *pthread\_setconcurrency()* and return a constant for *pthread\_getconcurrency()*. For this reason, it is  
51753 not recommended that portable applications use this function.

51754 **RATIONALE**

51755 None.

51756 **FUTURE DIRECTIONS**

51757 These functions may be removed in a future version.

51758 **SEE ALSO**51759 XBD [<pthread.h>](#)51760 **CHANGE HISTORY**

51761 First released in Issue 5.

51762 **Issue 7**

51763 SD5-XSH-ERN-184 is applied.

51764 The *pthread\_getconcurrency()* and *pthread\_setconcurrency()* functions are marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51765 **NAME**

51766 pthread\_getcpuclockid — access a thread CPU-time clock (ADVANCED REALTIME  
51767 THREADS)

51768 **SYNOPSIS**

```
51769 TCT #include <pthread.h>
51770 #include <time.h>
51771 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

51772 **DESCRIPTION**

51773 The *pthread\_getcpuclockid()* function shall return in *clock\_id* the clock ID of the CPU-time clock of  
51774 the thread specified by *thread\_id*, if the thread specified by *thread\_id* exists.

51775 **RETURN VALUE**

51776 Upon successful completion, *pthread\_getcpuclockid()* shall return zero; otherwise, an error  
51777 number shall be returned to indicate the error.

51778 **ERRORS**

51779 No errors are defined.

51780 **EXAMPLES**

51781 None.

51782 **APPLICATION USAGE**

51783 The *pthread\_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not  
51784 be provided on all implementations.

51785 **RATIONALE**

51786 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
51787 that the function should fail and report an [ESRCH] error.

51788 **FUTURE DIRECTIONS**

51789 None.

51790 **SEE ALSO**

51791 *clock\_getcpuclockid()*, *clock\_getres()*, *timer\_create()*

51792 XBD [<pthread.h>](#), [<time.h>](#)

51793 **CHANGE HISTORY**

51794 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

51795 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

51796 **Issue 7**

51797 The *pthread\_getcpuclockid()* function is moved from the Threads option.

51798 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread\_getschedparam()**

System Interfaces

51799 **NAME**

51800 pthread\_getschedparam, pthread\_setschedparam — dynamic thread scheduling parameters  
 51801 access (**REALTIME THREADS**)

51802 **SYNOPSIS**

```
51803 TPS #include <pthread.h>
51804 int pthread_getschedparam(pthread_t thread, int *restrict policy,
51805 struct sched_param *restrict param);
51806 int pthread_setschedparam(pthread_t thread, int policy,
51807 const struct sched_param *param);
```

51808 **DESCRIPTION**

51809 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall, respectively, get and set  
 51810 the scheduling policy and parameters of individual threads within a multi-threaded process to  
 51811 be retrieved and set. For SCHED\_FIFO and SCHED\_RR, the only required member of the  
 51812 **sched\_param** structure is the priority *sched\_priority*. For SCHED\_OTHER, the affected  
 51813 scheduling parameters are implementation-defined.

51814 The *pthread\_getschedparam()* function shall retrieve the scheduling policy and scheduling  
 51815 parameters for the thread whose thread ID is given by *thread* and shall store those values in  
 51816 *policy* and *param*, respectively. The priority value returned from *pthread\_getschedparam()* shall be  
 51817 the value specified by the most recent *pthread\_setschedparam()*, *pthread\_setschedprio()*, or  
 51818 *pthread\_create()* call affecting the target thread. It shall not reflect any temporary adjustments to  
 51819 its priority as a result of any priority inheritance or ceiling functions. The *pthread\_setschedparam()*  
 51820 function shall set the scheduling policy and associated scheduling parameters for the thread  
 51821 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*  
 51822 and *param*, respectively.

51823 The *policy* parameter may have the value SCHED\_OTHER, SCHED\_FIFO, or SCHED\_RR. The  
 51824 scheduling parameters for the SCHED\_OTHER policy are implementation-defined. The  
 51825 SCHED\_FIFO and SCHED\_RR policies shall have a single scheduling parameter, *priority*.

51826 TSP If **\_POSIX\_THREAD\_SPORADIC\_SERVER** is defined, then the *policy* argument may have the  
 51827 value SCHED\_SPORADIC, with the exception for the *pthread\_setschedparam()* function that if the  
 51828 scheduling policy was not SCHED\_SPORADIC at the time of the call, it is implementation-  
 51829 defined whether the function is supported; in other words, the implementation need not allow  
 51830 the application to dynamically change the scheduling policy to SCHED\_SPORADIC. The  
 51831 sporadic server scheduling policy has the associated parameters *sched\_ss\_low\_priority*,  
 51832 *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, *sched\_priority*, and *sched\_ss\_max\_repl*. The specified  
 51833 *sched\_ss\_repl\_period* shall be greater than or equal to the specified *sched\_ss\_init\_budget* for the  
 51834 function to succeed; if it is not, then the function shall fail. The value of *sched\_ss\_max\_repl* shall  
 51835 be within the inclusive range [1, {SS\_REPL\_MAX}] for the function to succeed; if not, the function  
 51836 shall fail. It is unspecified whether the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are  
 51837 stored as provided by this function or are rounded to align with the resolution of the clock being  
 51838 used.

51839 If the *pthread\_setschedparam()* function fails, the scheduling parameters shall not be changed for  
 51840 the target thread.

51841 **RETURN VALUE**

51842 If successful, the *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall return zero;  
 51843 otherwise, an error number shall be returned to indicate the error.

**51844 ERRORS**

51845 The *pthread\_setschedparam()* function may fail if:

51846 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated  
51847 with the scheduling policy *policy* is invalid.

51848 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an  
51849 unsupported value.

51850 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to  
51851 SCHED\_SPORADIC, and the implementation does not support this change.

51852 [EPERM] The caller does not have appropriate privileges to set either the scheduling  
51853 parameters or the scheduling policy of the specified thread.

51854 [EPERM] The implementation does not allow the application to modify one of the  
51855 parameters to the value specified.

51856 These functions shall not return an error code of [EINTR].

**51857 EXAMPLES**

51858 None.

**51859 APPLICATION USAGE**

51860 None.

**51861 RATIONALE**

51862 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
51863 that the function should fail and report an [ESRCH] error.

**51864 FUTURE DIRECTIONS**

51865 None.

**51866 SEE ALSO**

51867 *pthread\_setschedprio()*, *sched\_getparam()*, *sched\_getscheduler()*

51868 XBD <pthread.h>, <sched.h>

**51869 CHANGE HISTORY**

51870 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**51871 Issue 6**

51872 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked as part of the  
51873 Threads and Thread Execution Scheduling options.

51874 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
51875 implementation does not support the Thread Execution Scheduling option.

51876 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
51877 *pthread\_setschedparam()* function so that its second argument is of type **int**.

51878 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

51879 The **restrict** keyword is added to the *pthread\_getschedparam()* prototype for alignment with the  
51880 ISO/IEC 9899: 1999 standard.

51881 The Open Group Corrigendum U047/1 is applied.

51882 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a  
51883 call to the *pthread\_setschedprio()* function.

**pthread\_getschedparam()***System Interfaces*51884 **Issue 7**

51885 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are moved from the Threads  
51886 option.

51887 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
51888 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.

51889 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**51890 NAME**

51891 pthread\_getspecific, pthread\_setspecific — thread-specific data management

**51892 SYNOPSIS**

51893 #include <pthread.h>

51894 void \*pthread\_getspecific(pthread\_key\_t key);

51895 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

**51896 DESCRIPTION**

51897 The *pthread\_getspecific()* function shall return the value currently bound to the specified *key* on  
51898 behalf of the calling thread.

51899 The *pthread\_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a  
51900 previous call to *pthread\_key\_create()*. Different threads may bind different values to the same  
51901 key. These values are typically pointers to blocks of dynamically allocated memory that have  
51902 been reserved for use by the calling thread.

51903 The effect of calling *pthread\_getspecific()* or *pthread\_setspecific()* with a *key* value not obtained  
51904 from *pthread\_key\_create()* or after *key* has been deleted with *pthread\_key\_delete()* is undefined.

51905 Both *pthread\_getspecific()* and *pthread\_setspecific()* may be called from a thread-specific data  
51906 destructor function. A call to *pthread\_getspecific()* for the thread-specific data key being  
51907 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)  
51908 by a call to *pthread\_setspecific()*. Calling *pthread\_setspecific()* from a thread-specific data  
51909 destructor routine may result either in lost storage (after at least  
51910 PTHREAD\_DESTRUCTOR\_ITERATIONS attempts at destruction) or in an infinite loop.

51911 Both functions may be implemented as macros.

**51912 RETURN VALUE**

51913 The *pthread\_getspecific()* function shall return the thread-specific data value associated with the  
51914 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be  
51915 returned.

51916 If successful, the *pthread\_setspecific()* function shall return zero; otherwise, an error number shall  
51917 be returned to indicate the error.

**51918 ERRORS**

51919 No errors are returned from *pthread\_getspecific()*.

51920 The *pthread\_setspecific()* function shall fail if:

51921 [ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

51922 The *pthread\_setspecific()* function shall not return an error code of [EINTR].

**51923 EXAMPLES**

51924 None.

**51925 APPLICATION USAGE**

51926 None.

**51927 RATIONALE**

51928 Performance and ease-of-use of *pthread\_getspecific()* are critical for functions that rely on  
51929 maintaining state in thread-specific data. Since no errors are required to be detected by it, and  
51930 since the only error that could be detected is the use of an invalid key, the function to  
51931 *pthread\_getspecific()* has been designed to favor speed and simplicity over error reporting.

51932 If an implementation detects that the value specified by the *key* argument to *pthread\_setspecific()*  
51933 does not refer to a key value obtained from *pthread\_key\_create()* or refers to a key that has been

**pthread\_getspecific()**

System Interfaces

51934 deleted with *pthread\_key\_delete()*, it is recommended that the function should fail and report an  
51935 [EINVAL] error.

**FUTURE DIRECTIONS**

51937 None.

**SEE ALSO**

51939 *pthread\_key\_create()*

51940 XBD <pthread.h>

**CHANGE HISTORY**

51942 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

51944 The *pthread\_getspecific()* and *pthread\_setspecific()* functions are marked as part of the Threads  
51945 option.

51946 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

51947 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS  
51948 section so that the [ENOMEM] error case is changed from “to associate the value with the key”  
51949 to “to associate the non-NULL value with the key”.

**Issue 7**

51950 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

51952 The *pthread\_getspecific()* and *pthread\_setspecific()* functions are moved from the Threads option to  
51953 the Base.

51954 The [EINVAL] error for a key value not obtained from *pthread\_key\_create()* or a key deleted with  
51955 *pthread\_key\_delete()* is removed; this condition results in undefined behavior.

51956 **NAME**

51957 pthread\_join — wait for thread termination

51958 **SYNOPSIS**

51959 #include &lt;pthread.h&gt;

51960 int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

51961 **DESCRIPTION**

51962 The *pthread\_join()* function shall suspend execution of the calling thread until the target *thread*  
 51963 terminates, unless the target *thread* has already terminated. On return from a successful  
 51964 *pthread\_join()* call with a non-NULL *value\_ptr* argument, the value passed to *pthread\_exit()* by  
 51965 the terminating thread shall be made available in the location referenced by *value\_ptr*. When a  
 51966 *pthread\_join()* returns successfully, the target thread has been terminated. The results of multiple  
 51967 simultaneous calls to *pthread\_join()* specifying the same target thread are undefined. If the  
 51968 thread calling *pthread\_join()* is canceled, then the target thread shall not be detached.

51969 It is unspecified whether a thread that has exited but remains unjoined counts against  
 51970 {PTHREAD\_THREADS\_MAX}.

51971 The behavior is undefined if the value specified by the *thread* argument to *pthread\_join()* does not  
 51972 refer to a joinable thread.

51973 The behavior is undefined if the value specified by the *thread* argument to *pthread\_join()* refers to  
 51974 the calling thread.

51975 **RETURN VALUE**

51976 If successful, the *pthread\_join()* function shall return zero; otherwise, an error number shall be  
 51977 returned to indicate the error.

51978 **ERRORS**51979 The *pthread\_join()* function may fail if:

51980 [EDEADLK] A deadlock was detected.

51981 The *pthread\_join()* function shall not return an error code of [EINTR].51982 **EXAMPLES**

51983 An example of thread creation and deletion follows:

```

51984 typedef struct {
51985     int *ar;
51986     long n;
51987 } subarray;

51988 void *
51989 incesr(void *arg)
51990 {
51991     long i;

51992     for (i = 0; i < ((subarray *)arg)->n; i++)
51993         ((subarray *)arg)->ar[i]++;
51994 }

51995 int main(void)
51996 {
51997     int ar[1000000];
51998     pthread_t th1, th2;
51999     subarray sb1, sb2;

```

**pthread\_join()**

```

52000         sb1.ar = &ar[0];
52001         sb1.n = 500000;
52002         (void) pthread_create(&th1, NULL, incer, &sb1);

52003         sb2.ar = &ar[500000];
52004         sb2.n = 500000;
52005         (void) pthread_create(&th2, NULL, incer, &sb2);

52006         (void) pthread_join(th1, NULL);
52007         (void) pthread_join(th2, NULL);
52008         return 0;
52009     }

```

52010 **APPLICATION USAGE**

52011 None.

52012 **RATIONALE**

52013 The *pthread\_join()* function is a convenience that has proven useful in multi-threaded  
52014 applications. It is true that a programmer could simulate this function if it were not provided by  
52015 passing extra state as part of the argument to the *start\_routine()*. The terminating thread would  
52016 set a flag to indicate termination and broadcast a condition that is part of that state; a joining  
52017 thread would wait on that condition variable. While such a technique would allow a thread to  
52018 wait on more complex conditions (for example, waiting for multiple threads to terminate),  
52019 waiting on individual thread termination is considered widely useful. Also, including the  
52020 *pthread\_join()* function in no way precludes a programmer from coding such complex waits.  
52021 Thus, while not a primitive, including *pthread\_join()* in this volume of POSIX.1-2008 was  
52022 considered valuable.

52023 The *pthread\_join()* function provides a simple mechanism allowing an application to wait for a  
52024 thread to terminate. After the thread terminates, the application may then choose to clean up  
52025 resources that were used by the thread. For instance, after *pthread\_join()* returns, any  
52026 application-provided stack storage could be reclaimed.

52027 The *pthread\_join()* or *pthread\_detach()* function should eventually be called for every thread that  
52028 is created with the *detachstate* attribute set to `PTHREAD_CREATE_JOINABLE` so that storage  
52029 associated with the thread may be reclaimed.

52030 The interaction between *pthread\_join()* and cancellation is well-defined for the following reasons:

- 52031 • The *pthread\_join()* function, like all other non-async-cancel-safe functions, can only be  
52032 called with deferred cancelability type.
- 52033 • Cancellation cannot occur in the disabled cancelability state.

52034 Thus, only the default cancelability state need be considered. As specified, either the  
52035 *pthread\_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the  
52036 application, since either a cancellation handler is run or *pthread\_join()* returns. There are no race  
52037 conditions since *pthread\_join()* was called in the deferred cancelability state.

52038 If an implementation detects that the value specified by the *thread* argument to *pthread\_join()*  
52039 does not refer to a joinable thread, it is recommended that the function should fail and report an  
52040 `[EINVAL]` error.

52041 If an implementation detects that the value specified by the *thread* argument to *pthread\_join()*  
52042 refers to the calling thread, it is recommended that the function should fail and report an  
52043 `[EDEADLK]` error.

52044 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended

52045 that the function should fail and report an [ESRCH] error.

52046 **FUTURE DIRECTIONS**

52047 None.

52048 **SEE ALSO**

52049 *pthread\_create()*, *wait()*

52050 XBD Section 4.11 (on page 110), <pthread.h>

52051 **CHANGE HISTORY**

52052 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52053 **Issue 6**

52054 The *pthread\_join()* function is marked as part of the Threads option.

52055 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS  
52056 section so that the [EINVAL] error is made optional and the words “the implementation has  
52057 detected” are removed from it.

52058 **Issue 7**

52059 The *pthread\_join()* function is moved from the Threads option to the Base.

52060 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

52061 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined  
52062 behavior.

52063 The [EDEADLK] error for the calling thread is removed; this condition results in undefined  
52064 behavior.

**pthread\_key\_create()**52065 **NAME**

52066 pthread\_key\_create — thread-specific data key creation

52067 **SYNOPSIS**

52068 #include &lt;pthread.h&gt;

52069 int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

52070 **DESCRIPTION**

52071 The *pthread\_key\_create()* function shall create a thread-specific data key visible to all threads in  
 52072 the process. Key values provided by *pthread\_key\_create()* are opaque objects used to locate  
 52073 thread-specific data. Although the same key value may be used by different threads, the values  
 52074 bound to the key by *pthread\_setspecific()* are maintained on a per-thread basis and persist for the  
 52075 life of the calling thread.

52076 Upon key creation, the value NULL shall be associated with the new key in all active threads.  
 52077 Upon thread creation, the value NULL shall be associated with all defined keys in the new  
 52078 thread.

52079 An optional destructor function may be associated with each key value. At thread exit, if a key  
 52080 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with  
 52081 that key, the value of the key is set to NULL, and then the function pointed to is called with the  
 52082 previously associated value as its sole argument. The order of destructor calls is unspecified if  
 52083 more than one destructor exists for a thread when it exits.

52084 If, after all the destructors have been called for all non-NULL values with associated destructors,  
 52085 there are still some non-NULL values with associated destructors, then the process is repeated.  
 52086 If, after at least {PTHREAD\_DESTRUCTOR\_ITERATIONS} iterations of destructor calls for  
 52087 outstanding non-NULL values, there are still some non-NULL values with associated  
 52088 destructors, implementations may stop calling destructors, or they may continue calling  
 52089 destructors until no non-NULL values with associated destructors exist, even though this might  
 52090 result in an infinite loop.

52091 **RETURN VALUE**

52092 If successful, the *pthread\_key\_create()* function shall store the newly created key value at \*key and  
 52093 shall return zero. Otherwise, an error number shall be returned to indicate the error.

52094 **ERRORS**52095 The *pthread\_key\_create()* function shall fail if:

52096 [EAGAIN] The system lacked the necessary resources to create another thread-specific  
 52097 data key, or the system-imposed limit on the total number of keys per process  
 52098 {PTHREAD\_KEYS\_MAX} has been exceeded.

52099 [ENOMEM] Insufficient memory exists to create the key.

52100 The *pthread\_key\_create()* function shall not return an error code of [EINTR].52101 **EXAMPLES**

52102 The following example demonstrates a function that initializes a thread-specific data key when it  
 52103 is first called, and associates a thread-specific object with each calling thread, initializing this  
 52104 object when necessary.

```
52105 static pthread_key_t key;
52106 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

52107 static void
52108 make_key()
52109 {
```

```

52110         (void) pthread_key_create(&key, NULL);
52111     }
52112     func()
52113     {
52114         void *ptr;
52115         (void) pthread_once(&key_once, make_key);
52116         if ((ptr = pthread_getspecific(key)) == NULL) {
52117             ptr = malloc(OBJECT_SIZE);
52118             ...
52119             (void) pthread_setspecific(key, ptr);
52120         }
52121         ...
52122     }

```

52123 Note that the key has to be initialized before *pthread\_getspecific()* or *pthread\_setspecific()* can be  
52124 used. The *pthread\_key\_create()* call could either be explicitly made in a module initialization  
52125 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt  
52126 to use the key before it is initialized is a programming error, making the code below incorrect.

```

52127     static pthread_key_t key;
52128     func()
52129     {
52130         void *ptr;
52131         /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
52132         if ((ptr = pthread_getspecific(key)) == NULL &&
52133             pthread_setspecific(key, NULL) != 0) {
52134             pthread_key_create(&key, NULL);
52135             ...
52136         }
52137     }

```

#### 52138 APPLICATION USAGE

52139 None.

#### 52140 RATIONALE

##### 52141 Destructor Functions

52142 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage  
52143 allocated dynamically on behalf of the calling thread. The destructor functions specified with  
52144 *pthread\_key\_create()* are intended to be used to free this storage when the thread exits. Thread  
52145 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may  
52146 persist outside the lexical scope in which the cancellation cleanup handlers operate.

52147 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
52148 be necessary to release the storage associated with the old value before the new value is bound.  
52149 Although the *pthread\_setspecific()* function could do this automatically, this feature is not needed  
52150 often enough to justify the added complexity. Instead, the programmer is responsible for freeing  
52151 the stale storage:

```

52152     pthread_getspecific(key, &old);
52153     new = allocate();
52154     destructor(old);

```

**pthread\_key\_create()**

52155 `pthread_setspecific(key, new);`

52156 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such  
52157 problems occur in the default cancellation state if no cancellation points occur between the get  
52158 and set.

52159 There is no notion of a destructor-safe function. If an application does not call `pthread_exit()`  
52160 from a signal handler, or if it blocks any signal whose handler may call `pthread_exit()` while  
52161 calling async-unsafe functions, all functions may be safely called from destructors.

### 52162 **Non-Idempotent Data Key Creation**

52163 There were requests to make `pthread_key_create()` idempotent with respect to a given `key` address  
52164 parameter. This would allow applications to call `pthread_key_create()` multiple times for a given  
52165 `key` address and be guaranteed that only one key would be created. Doing so would require the  
52166 key value to be previously initialized (possibly at compile time) to a known null value and  
52167 would require that implicit mutual-exclusion be performed based on the address and contents of  
52168 the `key` parameter in order to guarantee that exactly one key would be created.

52169 Unfortunately, the implicit mutual-exclusion would not be limited to only `pthread_key_create()`.  
52170 On many implementations, implicit mutual-exclusion would also have to be performed by  
52171 `pthread_getspecific()` and `pthread_setspecific()` in order to guard against using incompletely stored  
52172 or not-yet-visible key values. This could significantly increase the cost of important operations,  
52173 particularly `pthread_getspecific()`.

52174 Thus, this proposal was rejected. The `pthread_key_create()` function performs no implicit  
52175 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once  
52176 per key before use of the key. Several straightforward mechanisms can already be used to  
52177 accomplish this, including calling explicit module initialization functions, using mutexes, and  
52178 using `pthread_once()`. This places no significant burden on the programmer, introduces no  
52179 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows  
52180 commonly used thread-specific data operations to be more efficient.

### 52181 **FUTURE DIRECTIONS**

52182 None.

### 52183 **SEE ALSO**

52184 [pthread\\_getspecific\(\)](#), [pthread\\_key\\_delete\(\)](#)

52185 XBD [<pthread.h>](#)

### 52186 **CHANGE HISTORY**

52187 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 52188 **Issue 6**

52189 The `pthread_key_create()` function is marked as part of the Threads option.

52190 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

#### 52191 **Issue 7**

52192 The `pthread_key_create()` function is moved from the Threads option to the Base.

52193 **NAME**

52194 pthread\_key\_delete — thread-specific data key deletion

52195 **SYNOPSIS**

52196 #include &lt;pthread.h&gt;

52197 int pthread\_key\_delete(pthread\_key\_t key);

52198 **DESCRIPTION**

52199 The *pthread\_key\_delete()* function shall delete a thread-specific data key previously returned by  
 52200 *pthread\_key\_create()*. The thread-specific data values associated with *key* need not be NULL at  
 52201 the time *pthread\_key\_delete()* is called. It is the responsibility of the application to free any  
 52202 application storage or perform any cleanup actions for data structures related to the deleted key  
 52203 or associated thread-specific data in any threads; this cleanup can be done either before or after  
 52204 *pthread\_key\_delete()* is called. Any attempt to use *key* following the call to *pthread\_key\_delete()*  
 52205 results in undefined behavior.

52206 The *pthread\_key\_delete()* function shall be callable from within destructor functions. No  
 52207 destructor functions shall be invoked by *pthread\_key\_delete()*. Any destructor function that may  
 52208 have been associated with *key* shall no longer be called upon thread exit.

52209 **RETURN VALUE**

52210 If successful, the *pthread\_key\_delete()* function shall return zero; otherwise, an error number shall  
 52211 be returned to indicate the error.

52212 **ERRORS**52213 The *pthread\_key\_delete()* function shall not return an error code of [EINTR].52214 **EXAMPLES**

52215 None.

52216 **APPLICATION USAGE**

52217 None.

52218 **RATIONALE**

52219 A thread-specific data key deletion function has been included in order to allow the resources  
 52220 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
 52221 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
 52222 unloaded.

52223 Conforming applications are responsible for performing any cleanup actions needed for data  
 52224 structures associated with the key to be deleted, including data referenced by thread-specific  
 52225 data values. No such cleanup is done by *pthread\_key\_delete()*. In particular, destructor functions  
 52226 are not called. There are several reasons for this division of responsibility:

- 52227 1. The associated destructor functions used to free thread-specific data at thread exit time  
 52228 are only guaranteed to work correctly when called in the thread that allocated the thread-  
 52229 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot  
 52230 be used to free thread-specific data in other threads at key deletion time. Attempting to  
 52231 have them called by other threads at key deletion time would require other threads to be  
 52232 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,  
 52233 including holding locks necessary for the destructor to run, this approach would fail. In  
 52234 general, there is no safe mechanism whereby an implementation could free thread-  
 52235 specific data at key deletion time.
- 52236 2. Even if there were a means of safely freeing thread-specific data associated with keys to  
 52237 be deleted, doing so would require that implementations be able to enumerate the  
 52238 threads with non-NULL data and potentially keep them from creating more thread-

**pthread\_key\_delete()**

System Interfaces

- 52239 specific data while the key deletion is occurring. This special case could cause extra  
52240 synchronization in the normal case, which would otherwise be unnecessary.
- 52241 For an application to know that it is safe to delete a key, it has to know that all the threads that  
52242 might potentially ever use the key do not attempt to use it again. For example, it could know  
52243 this if all the client threads have called a cleanup procedure declaring that they are through with  
52244 the module that is being shut down, perhaps by setting a reference count to zero.
- 52245 If an implementation detects that the value specified by the *key* argument to *pthread\_key\_delete()*  
52246 does not refer to a key value obtained from *pthread\_key\_create()* or refers to a key that has been  
52247 deleted with *pthread\_key\_delete()*, it is recommended that the function should fail and report an  
52248 [EINVAL] error.
- 52249 **FUTURE DIRECTIONS**
- 52250 None.
- 52251 **SEE ALSO**
- 52252 [pthread\\_key\\_create\(\)](#)
- 52253 XBD [<pthread.h>](#)
- 52254 **CHANGE HISTORY**
- 52255 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 52256 **Issue 6**
- 52257 The *pthread\_key\_delete()* function is marked as part of the Threads option.
- 52258 **Issue 7**
- 52259 The *pthread\_key\_delete()* function is moved from the Threads option to the Base.
- 52260 The [EINVAL] error for a key value not obtained from *pthread\_key\_create()* or a key deleted with  
52261 *pthread\_key\_delete()* is removed; this condition results in undefined behavior.

52262 **NAME**

52263 pthread\_kill — send a signal to a thread

52264 **SYNOPSIS**

```
52265 CX #include <signal.h>
52266 int pthread_kill(pthread_t thread, int sig);
```

52267 **DESCRIPTION**52268 The *pthread\_kill()* function shall request that a signal be delivered to the specified thread.52269 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.52270 **RETURN VALUE**52271 Upon successful completion, the function shall return a value of zero. Otherwise, the function  
52272 shall return an error number. If the *pthread\_kill()* function fails, no signal shall be sent.52273 **ERRORS**52274 The *pthread\_kill()* function shall fail if:52275 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.52276 The *pthread\_kill()* function shall not return an error code of [EINTR].52277 **EXAMPLES**

52278 None.

52279 **APPLICATION USAGE**52280 The *pthread\_kill()* function provides a mechanism for asynchronously directing a signal at a  
52281 thread in the calling process. This could be used, for example, by one thread to affect broadcast  
52282 delivery of a signal to a set of threads.52283 Note that *pthread\_kill()* only causes the signal to be handled in the context of the given thread;  
52284 the signal action (termination or stopping) affects the process as a whole.52285 **RATIONALE**52286 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
52287 that the function should fail and report an [ESRCH] error.52288 **FUTURE DIRECTIONS**

52289 None.

52290 **SEE ALSO**52291 *kill()*, *pthread\_self()*, *raise()*

52292 XBD &lt;signal.h&gt;

52293 **CHANGE HISTORY**

52294 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52295 **Issue 6**52296 The *pthread\_kill()* function is marked as part of the Threads option.

52297 The APPLICATION USAGE section is added.

52298 **Issue 7**52299 The *pthread\_kill()* function is moved from the Threads option to the Base.

52300 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread\_mutex\_consistent()**52301 **NAME**

52302 pthread\_mutex\_consistent — mark state protected by robust mutex as consistent

52303 **SYNOPSIS**

52304 #include &lt;pthread.h&gt;

52305 int pthread\_mutex\_consistent(pthread\_mutex\_t \*mutex);

52306 **DESCRIPTION**52307 If *mutex* is a robust mutex in an inconsistent state, the *pthread\_mutex\_consistent()* function can be  
52308 used to mark the state protected by the mutex referenced by *mutex* as consistent again.52309 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes  
52310 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the  
52311 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again  
52312 until the state is marked consistent.52313 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates  
52314 before calling either *pthread\_mutex\_consistent()* or *pthread\_mutex\_unlock()*, the next thread that  
52315 acquires the mutex lock shall be notified about the state of the mutex by the return value  
52316 [EOWNERDEAD].52317 The behavior is undefined if the value specified by the *mutex* argument to  
52318 *pthread\_mutex\_consistent()* does not refer to an initialized mutex.52319 **RETURN VALUE**52320 Upon successful completion, the *pthread\_mutex\_consistent()* function shall return zero.  
52321 Otherwise, an error value shall be returned to indicate the error.52322 **ERRORS**52323 The *pthread\_mutex\_consistent()* function shall fail if:52324 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an  
52325 inconsistent state.

52326 These functions shall not return an error code of [EINTR].

52327 **EXAMPLES**

52328 None.

52329 **APPLICATION USAGE**52330 The *pthread\_mutex\_consistent()* function is only responsible for notifying the implementation that  
52331 the state protected by the mutex has been recovered and that normal operations with the mutex  
52332 can be resumed. It is the responsibility of the application to recover the state so it can be reused.  
52333 If the application is not able to perform the recovery, it can notify the implementation that the  
52334 situation is unrecoverable by a call to *pthread\_mutex\_unlock()* without a prior call to  
52335 *pthread\_mutex\_consistent()*, in which case subsequent threads that attempt to lock the mutex will  
52336 fail to acquire the lock and be returned [ENOTRECOVERABLE].52337 **RATIONALE**52338 If an implementation detects that the value specified by the *mutex* argument to  
52339 *pthread\_mutex\_consistent()* does not refer to an initialized mutex, it is recommended that the  
52340 function should fail and report an [EINVAL] error.52341 **FUTURE DIRECTIONS**

52342 None.

- 52343 **SEE ALSO**
- 52344 [pthread\\_mutex\\_lock\(\)](#), [pthread\\_mutexattr\\_getrobust\(\)](#)
- 52345 XBD [<pthread.h>](#)
- 52346 **CHANGE HISTORY**
- 52347 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutex\_destroy()**

System Interfaces

52348 **NAME**

52349 pthread\_mutex\_destroy, pthread\_mutex\_init — destroy and initialize a mutex

52350 **SYNOPSIS**

```
52351 #include <pthread.h>
52352
52352 int pthread_mutex_destroy(pthread_mutex_t *mutex);
52353 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
52354     const pthread_mutexattr_t *restrict attr);
52355 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

52356 **DESCRIPTION**

52357 The *pthread\_mutex\_destroy()* function shall destroy the mutex object referenced by *mutex*; the  
 52358 mutex object becomes, in effect, uninitialized. An implementation may cause  
 52359 *pthread\_mutex\_destroy()* to set the object referenced by *mutex* to an invalid value.

52360 A destroyed mutex object can be reinitialized using *pthread\_mutex\_init()*; the results of otherwise  
 52361 referencing the object after it has been destroyed are undefined.

52362 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked  
 52363 mutex or a mutex that is referenced (for example, while being used in a *pthread\_cond\_timedwait()*  
 52364 or *pthread\_cond\_wait()*) by another thread results in undefined behavior.

52365 The *pthread\_mutex\_init()* function shall initialize the mutex referenced by *mutex* with attributes  
 52366 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the  
 52367 same as passing the address of a default mutex attributes object. Upon successful initialization,  
 52368 the state of the mutex becomes initialized and unlocked.

52369 Only *mutex* itself may be used for performing synchronization. The result of referring to copies  
 52370 of *mutex* in calls to *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *pthread\_mutex\_unlock()*, and  
 52371 *pthread\_mutex\_destroy()* is undefined.

52372 Attempting to initialize an already initialized mutex results in undefined behavior.

52373 In cases where default mutex attributes are appropriate, the macro  
 52374 PTHREAD\_MUTEX\_INITIALIZER can be used to initialize mutexes that are statically allocated.  
 52375 The effect shall be equivalent to dynamic initialization by a call to *pthread\_mutex\_init()* with  
 52376 parameter *attr* specified as NULL, except that no error checks are performed.

52377 The behavior is undefined if the value specified by the *mutex* argument to  
 52378 *pthread\_mutex\_destroy()* does not refer to an initialized mutex.

52379 The behavior is undefined if the value specified by the *attr* argument to *pthread\_mutex\_init()*  
 52380 does not refer to an initialized mutex attributes object.

52381 **RETURN VALUE**

52382 If successful, the *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions shall return zero;  
 52383 otherwise, an error number shall be returned to indicate the error.

52384 **ERRORS**

52385 The *pthread\_mutex\_init()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 52386 | [EAGAIN] | The system lacked the necessary resources (other than memory) to initialize another mutex. |
| 52387 |          |  |
| 52388 | [ENOMEM] | Insufficient memory exists to initialize the mutex.  |
| 52389 | [EPERM]  | The caller does not have the privilege to perform the operation.                           |

52390 The *pthread\_mutex\_init()* function may fail if:

52391 [EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set  
52392 without the process-shared attribute being set.

52393 These functions shall not return an error code of [EINTR].

#### 52394 EXAMPLES

52395 None.

#### 52396 APPLICATION USAGE

52397 None.

#### 52398 RATIONALE

52399 If an implementation detects that the value specified by the *mutex* argument to  
52400 *pthread\_mutex\_destroy()* does not refer to an initialized mutex, it is recommended that the  
52401 function should fail and report an [EINVAL] error.

52402 If an implementation detects that the value specified by the *mutex* argument to  
52403 *pthread\_mutex\_destroy()* or *pthread\_mutex\_init()* refers to a locked mutex or a mutex that is  
52404 referenced (for example, while being used in a *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*)  
52405 by another thread, or detects that the value specified by the *mutex* argument to  
52406 *pthread\_mutex\_init()* refers to an already initialized mutex, it is recommended that the function  
52407 should fail and report an [EBUSY] error.

52408 If an implementation detects that the value specified by the *attr* argument to  
52409 *pthread\_mutex\_init()* does not refer to an initialized mutex attributes object, it is recommended  
52410 that the function should fail and report an [EINVAL] error.

#### 52411 Alternate Implementations Possible

52412 This volume of POSIX.1-2008 supports several alternative implementations of mutexes. An  
52413 implementation may store the lock directly in the object of type **pthread\_mutex\_t**. Alternatively,  
52414 an implementation may store the lock in the heap and merely store a pointer, handle, or unique  
52415 ID in the mutex object. Either implementation has advantages or may be required on certain  
52416 hardware configurations. So that portable code can be written that is invariant to this choice, this  
52417 volume of POSIX.1-2008 does not define assignment or equality for this type, and it uses the  
52418 term “initialize” to reinforce the (more restrictive) notion that the lock may actually reside in the  
52419 mutex object itself.

52420 Note that this precludes an over-specification of the type of the mutex or condition variable and  
52421 motivates the opaqueness of the type.

52422 An implementation is permitted, but not required, to have *pthread\_mutex\_destroy()* store an  
52423 illegal value into the mutex. This may help detect erroneous programs that try to lock (or  
52424 otherwise reference) a mutex that has already been destroyed.

#### 52425 Tradeoff Between Error Checks and Performance Supported

52426 Many error conditions that can occur are not required to be detected by the implementation in  
52427 order to let implementations trade off performance *versus* degree of error checking according to  
52428 the needs of their specific applications and execution environment. As a general rule, conditions  
52429 caused by the system (such as insufficient memory) are required to be detected, but conditions  
52430 caused by an erroneously coded application (such as failing to provide adequate  
52431 synchronization to prevent a mutex from being deleted while in use) are specified to result in  
52432 undefined behavior.

52433 A wide range of implementations is thus made possible. For example, an implementation

**pthread\_mutex\_destroy()**

intended for application debugging may implement all of the error checks, but an implementation running a single, provably correct application under very tight performance constraints in an embedded computer might implement minimal checks. An implementation might even be provided in two versions, similar to the options that compilers provide: a full-checking, but slower version; and a limited-checking, but faster version. To forbid this optionality would be a disservice to users.

By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly coded) application might do, and by defining that resource-not-available errors are mandatory, this volume of POSIX.1-2008 ensures that a fully-conforming application is portable across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does. When the behavior is undefined, no error number is specified to be returned on implementations that do detect the condition. This is because undefined behavior means *anything* can happen, which includes returning with any value (which might happen to be a valid, but different, error number). However, since the error number might be useful to application developers when diagnosing problems during application development, a recommendation is made in rationale that implementors should return a particular error number if their implementation does detect the condition.

**Why No Limits are Defined**

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

**Static Initializers for Mutexes and Condition Variables**

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo()* might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
    /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```

52479 void foo()
52480 {
52481     pthread_mutex_lock(&foo_mutex);
52482     /* Do work. */
52483     pthread_mutex_unlock(&foo_mutex);
52484 }

```

52485 Note that the static initialization both eliminates the need for the initialization test inside  
52486 *pthread\_once()* and the fetch of *&foo\_mutex* to learn the address to be passed to  
52487 *pthread\_mutex\_lock()* or *pthread\_mutex\_unlock()*.

52488 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a  
52489 large class of systems; those where the (entire) synchronization object can be stored in  
52490 application memory.

52491 Yet the locking performance question is likely to be raised for machines that require mutexes to  
52492 be allocated out of special memory. Such machines actually have to have mutexes and possibly  
52493 condition variables contain pointers to the actual hardware locks. For static initialization to work  
52494 on such machines, *pthread\_mutex\_lock()* also has to test whether or not the pointer to the actual  
52495 lock has been allocated. If it has not, *pthread\_mutex\_lock()* has to initialize it before use. The  
52496 reservation of such resources can be made when the program is loaded, and hence return codes  
52497 have not been added to mutex locking and condition-variable waiting to indicate failure to  
52498 complete initialization.

52499 This runtime test in *pthread\_mutex\_lock()* would at first seem to be extra work; an extra test is  
52500 required to see whether the pointer has been initialized. On most machines this would actually  
52501 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the  
52502 pointer if it has already been initialized. While the test might seem to add extra work, the extra  
52503 effort of testing a register is usually negligible since no extra memory references are actually  
52504 done. As more and more machines provide caches, the real expenses are memory references, not  
52505 instructions executed.

52506 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*  
52507 overhead in the most important case: on the lock operations that occur *after* the lock has been  
52508 initialized. This can be done by shifting more overhead to the less frequent operation:  
52509 initialization. Since out-of-line mutex allocation also means that an address has to be  
52510 dereferenced to find the actual lock, one technique that is widely applicable is to have static  
52511 initialization store a bogus value for that address; in particular, an address that causes a machine  
52512 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity  
52513 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent  
52514 lock operations incur no extra overhead since they do not “fault”. This is merely one technique  
52515 that can be used to support static initialization, while not adversely affecting the performance of  
52516 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

52517 The locking overhead for machines doing out-of-line mutex allocation is thus similar for  
52518 modules being implicitly initialized, where it is improved for those doing mutex allocation  
52519 entirely inline. The inline case is thus made much faster, and the out-of-line case is not  
52520 significantly worse.

52521 Besides the issue of locking performance for such machines, a concern is raised that it is possible  
52522 that threads would serialize contending for initialization locks when attempting to finish  
52523 initializing statically allocated mutexes. (Such finishing would typically involve taking an  
52524 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing  
52525 the internal lock.) First, many implementations would reduce such serialization by hashing on  
52526 the mutex address. Second, such serialization can only occur a bounded number of times. In

**pthread\_mutex\_destroy()**

52527 particular, it can happen at most as many times as there are statically allocated synchronization  
 52528 objects. Dynamically allocated objects would still be initialized via *pthread\_mutex\_init()* or  
 52529 *pthread\_cond\_init()*.

52530 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient  
 52531 performance for an application on some implementation, the application can avoid static  
 52532 initialization altogether by explicitly initializing all synchronization objects with the  
 52533 corresponding *pthread\_\*\_init()* functions, which are supported by all implementations. An  
 52534 implementation can also document the tradeoffs and advise which initialization technique is  
 52535 more efficient for that particular implementation.

**52536 Destroying Mutexes**

52537 A mutex can be destroyed immediately after it is unlocked. For example, consider the following  
 52538 code:

```
52539 struct obj {
52540 pthread_mutex_t om;
52541     int refcnt;
52542     ...
52543 };
52544 obj_done(struct obj *op)
52545 {
52546     pthread_mutex_lock(&op->om);
52547     if (--op->refcnt == 0) {
52548         pthread_mutex_unlock(&op->om);
52549 (A)     pthread_mutex_destroy(&op->om);
52550 (B)     free(op);
52551     } else
52552 (C)     pthread_mutex_unlock(&op->om);
52553 }
```

52554 In this case *obj* is reference counted and *obj\_done()* is called whenever a reference to the object is  
 52555 dropped. Implementations are required to allow an object to be destroyed and freed and  
 52556 potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line  
 52557 C).

**52558 Robust Mutexes**

52559 Implementations are required to provide robust mutexes for mutexes with the process-shared  
 52560 attribute set to *PTHREAD\_PROCESS\_SHARED*. Implementations are allowed, but not required,  
 52561 to provide robust mutexes when the process-shared attribute is set to  
 52562 *PTHREAD\_PROCESS\_PRIVATE*.

**52563 FUTURE DIRECTIONS**

52564 None.

**52565 SEE ALSO**

52566 *pthread\_mutex\_getprioceiling()*, *pthread\_mutexattr\_getrobust()*, *pthread\_mutex\_lock()*,  
 52567 *pthread\_mutex\_timedlock()*, *pthread\_mutexattr\_getshared()*

52568 XBD <pthread.h>

**52569 CHANGE HISTORY**

52570 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**52571 Issue 6**

52572 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are marked as part of the  
52573 Threads option.

52574 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
52575 IEEE Std 1003.1d-1999.

52576 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

52577 The **restrict** keyword is added to the *pthread\_mutex\_init()* prototype for alignment with the  
52578 ISO/IEC 9899: 1999 standard.

**52579 Issue 7**

52580 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

52581 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are moved from the Threads  
52582 option to the Base.

52583 The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is  
52584 removed; this condition results in undefined behavior.

52585 The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex  
52586 is removed; this condition results in undefined behavior.

**pthread\_mutex\_getprioceiling()**

System Interfaces

52587 **NAME**

52588 pthread\_mutex\_getprioceiling, pthread\_mutex\_setprioceiling — get and set the priority ceiling  
 52589 of a mutex (**REALTIME THREADS**)

52590 **SYNOPSIS**

```
52591 RPP|TPP #include <pthread.h>
52592
52592 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
52593 int *restrict prioceiling);
52594 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
52595 int prioceiling, int *restrict old_ceiling);
```

52596 **DESCRIPTION**

52597 The *pthread\_mutex\_getprioceiling()* function shall return the current priority ceiling of the mutex.

52598 The *pthread\_mutex\_setprioceiling()* function shall attempt to lock the mutex as if by a call to  
 52599 *pthread\_mutex\_lock()*, except that the process of locking the mutex need not adhere to the priority  
 52600 protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then  
 52601 release the mutex as if by a call to *pthread\_mutex\_unlock()*. When the change is successful, the  
 52602 previous value of the priority ceiling shall be returned in *old\_ceiling*.

52603 If the *pthread\_mutex\_setprioceiling()* function fails, the mutex priority ceiling shall not be  
 52604 changed.

52605 **RETURN VALUE**

52606 If successful, the *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions shall  
 52607 return zero; otherwise, an error number shall be returned to indicate the error.

52608 **ERRORS**

52609 These functions shall fail if:

52610 [EINVAL] The protocol attribute of *mutex* is PTHREAD\_PRIO\_NONE.

52611 [EPERM] The implementation requires appropriate privileges to perform the operation  
 52612 and the caller does not have appropriate privileges.

52613 The *pthread\_mutex\_setprioceiling()* function shall fail if:

52614 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
 52615 locks for *mutex* has been exceeded.

52616 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
 52617 thread already owns the mutex.

52618 [EINVAL] The mutex was created with the protocol attribute having the value  
 52619 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
 52620 the mutex's current priority ceiling, and the implementation adheres to the  
 52621 priority protect protocol in the process of locking the mutex.

52622 [ENOTRECOVERABLE]

52623 The mutex is a robust mutex and the state protected by the mutex is not  
 52624 recoverable.

52625 [EOWNERDEAD]

52626 The mutex is a robust mutex and the process containing the previous owning  
 52627 thread terminated while holding the mutex lock. The mutex lock shall be  
 52628 acquired by the calling thread and it is up to the new owner to make the state  
 52629 consistent (see *pthread\_mutex\_lock()*).

- 52630 The *pthread\_mutex\_setprioceiling()* function may fail if:
- 52631 [EDEADLK] A deadlock condition was detected.
- 52632 [EINVAL] The priority requested by *prioceiling* is out of range.
- 52633 [EOWNERDEAD]
- 52634 The mutex is a robust mutex and the previous owning thread terminated
- 52635 while holding the mutex lock. The mutex lock shall be acquired by the calling
- 52636 thread and it is up to the new owner to make the state consistent (see
- 52637 *pthread\_mutex\_lock()*).
- 52638 These functions shall not return an error code of [EINTR].
- 52639 **EXAMPLES**
- 52640 None.
- 52641 **APPLICATION USAGE**
- 52642 None.
- 52643 **RATIONALE**
- 52644 None.
- 52645 **FUTURE DIRECTIONS**
- 52646 None.
- 52647 **SEE ALSO**
- 52648 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*
- 52649 XBD <pthread.h>
- 52650 **CHANGE HISTORY**
- 52651 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 52652 Marked as part of the Realtime Threads Feature Group.
- 52653 **Issue 6**
- 52654 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are marked as
- 52655 part of the Threads and Thread Priority Protection options.
- 52656 The [ENOSYS] error conditions have been removed.
- 52657 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with
- 52658 IEEE Std 1003.1d-1999.
- 52659 The **restrict** keyword is added to the *pthread\_mutex\_getprioceiling()* and
- 52660 *pthread\_mutex\_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.
- 52661 **Issue 7**
- 52662 SD5-XSH-ERN-39 is applied.
- 52663 Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a “may fail”
- 52664 error.
- 52665 SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a “shall fail” error case
- 52666 for when the protocol attribute of *mutex* is PTHREAD\_PRIO\_NONE.
- 52667 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are moved from
- 52668 the Threads option to require support of either the Robust Mutex Priority Protection option or
- 52669 the Non-Robust Mutex Priority Protection option.

## **pthread\_mutex\_getprioceiling()**

*System Interfaces*

52670  
52671

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

52672 **NAME**

52673 pthread\_mutex\_init — destroy and initialize a mutex

52674 **SYNOPSIS**

```
52675 #include <pthread.h>
52676 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
52677 const pthread_mutexattr_t *restrict attr);
52678 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

52679 **DESCRIPTION**52680 Refer to [pthread\\_mutex\\_destroy\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutex\_lock()**

System Interfaces

52681 **NAME**

52682 pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a  
52683 mutex

52684 **SYNOPSIS**

```
52685 #include <pthread.h>
52686
52686 int pthread_mutex_lock(pthread_mutex_t *mutex);
52687 int pthread_mutex_trylock(pthread_mutex_t *mutex);
52688 int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

52689 **DESCRIPTION**

52690 The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock()*. If the  
52691 mutex is already locked, the calling thread shall block until the mutex becomes available. This  
52692 operation shall return with the mutex object referenced by *mutex* in the locked state with the  
52693 calling thread as its owner.

52694 If the mutex type is PTHREAD\_MUTEX\_NORMAL, deadlock detection shall not be provided.  
52695 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it  
52696 has not locked or a mutex which is unlocked, undefined behavior results.

52697 If the mutex type is PTHREAD\_MUTEX\_ERRORCHECK, then error checking shall be provided.  
52698 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a  
52699 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error  
52700 shall be returned.

52701 If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, then the mutex shall maintain the  
52702 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock  
52703 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be  
52704 incremented by one. Each time the thread unlocks the mutex, the lock count shall be  
52705 decremented by one. When the lock count reaches zero, the mutex shall become available for  
52706 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex  
52707 which is unlocked, an error shall be returned.

52708 If the mutex type is PTHREAD\_MUTEX\_DEFAULT, attempting to recursively lock the mutex  
52709 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling  
52710 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in  
52711 undefined behavior.

52712 The *pthread\_mutex\_trylock()* function shall be equivalent to *pthread\_mutex\_lock()*, except that if  
52713 the mutex object referenced by *mutex* is currently locked (by any thread, including the current  
52714 thread), the call shall return immediately. If the mutex type is PTHREAD\_MUTEX\_RECURSIVE  
52715 and the mutex is currently owned by the calling thread, the mutex lock count shall be  
52716 incremented by one and the *pthread\_mutex\_trylock()* function shall immediately return success.

52717 The *pthread\_mutex\_unlock()* function shall release the mutex object referenced by *mutex*. The  
52718 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are  
52719 threads blocked on the mutex object referenced by *mutex* when *pthread\_mutex\_unlock()* is called,  
52720 resulting in the mutex becoming available, the scheduling policy shall determine which thread  
52721 shall acquire the mutex.

52722 (In the case of PTHREAD\_MUTEX\_RECURSIVE mutexes, the mutex shall become available  
52723 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

52724 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the  
52725 thread shall resume waiting for the mutex as if it was not interrupted.

52726 If *mutex* is a robust mutex and the process containing the owning thread terminated while

52727 holding the mutex lock, a call to *pthread\_mutex\_lock()* shall return the error value  
 52728 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding  
 52729 the mutex lock, a call to *pthread\_mutex\_lock()* may return the error value [EOWNERDEAD] even  
 52730 if the process in which the owning thread resides has not terminated. In these cases, the mutex is  
 52731 locked by the thread but the state it protects is marked as inconsistent. The application should  
 52732 ensure that the state is made consistent for reuse and when that is complete call  
 52733 *pthread\_mutex\_consistent()*. If the application is unable to recover the state, it should unlock the  
 52734 mutex without a prior call to *pthread\_mutex\_consistent()*, after which the mutex is marked  
 52735 permanently unusable.

52736 If *mutex* does not refer to an initialized mutex object, the behavior of *pthread\_mutex\_lock()*,  
 52737 *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* is undefined.

#### 52738 RETURN VALUE

52739 If successful, the *pthread\_mutex\_lock()* and *pthread\_mutex\_unlock()* functions shall return zero;  
 52740 otherwise, an error number shall be returned to indicate the error.

52741 The *pthread\_mutex\_trylock()* function shall return zero if a lock on the mutex object referenced by  
 52742 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

#### 52743 ERRORS

52744 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions shall fail if:

52745 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
 52746 locks for *mutex* has been exceeded.

52747 RPP|TPP [EINVAL] The *mutex* was created with the protocol attribute having the value  
 52748 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
 52749 the mutex's current priority ceiling.

52750 [ENOTRECOVERABLE]

52751 The state protected by the mutex is not recoverable.

52752 [EOWNERDEAD]

52753 The mutex is a robust mutex and the process containing the previous owning  
 52754 thread terminated while holding the mutex lock. The mutex lock shall be  
 52755 acquired by the calling thread and it is up to the new owner to make the state  
 52756 consistent.

52757 The *pthread\_mutex\_lock()* function shall fail if:

52758 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
 52759 thread already owns the mutex.

52760 The *pthread\_mutex\_trylock()* function shall fail if:

52761 [EBUSY] The *mutex* could not be acquired because it was already locked.

52762 The *pthread\_mutex\_unlock()* function shall fail if:

52763 [EPERM] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK or the mutex is a  
 52764 robust mutex, and the current thread does not own the mutex.

52765 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions may fail if:

52766 [EOWNERDEAD]

52767 The mutex is a robust mutex and the previous owning thread terminated  
 52768 while holding the mutex lock. The mutex lock shall be acquired by the calling  
 52769 thread and it is up to the new owner to make the state consistent.

**pthread\_mutex\_lock()**

- 52770 The *pthread\_mutex\_lock()* function may fail if:
- 52771 [EDEADLK] A deadlock condition was detected.
- 52772 These functions shall not return an error code of [EINTR].

**EXAMPLES**

52773 None.

**APPLICATION USAGE**

52776 Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes it should check all return values for error conditions and if necessary take appropriate action.

**RATIONALE**

52782 Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the implementation of mutexes should be as efficient as possible, and this has ramifications on the features available at the interface.

52786 The mutex functions and the particular default settings of the mutex attributes have been motivated by the desire to not preclude fast, inlined implementations of mutex locking and unlocking.

52789 Since most attributes only need to be checked when a thread is going to be blocked, the use of attributes does not slow the (common) mutex-locking case.

52791 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a *mutex\_tryunlock* operation.

52794 For further rationale on the extended mutex types, see XRAT [Threads Extensions](#) (on page 3573).

52795 If an implementation detects that the value specified by the *mutex* argument does not refer to an initialized mutex object, it is recommended that the function should fail and report an [EINVAL] error.

**FUTURE DIRECTIONS**

52798 None.

**SEE ALSO**

52801 [pthread\\_mutex\\_consistent\(\)](#), [pthread\\_mutex\\_destroy\(\)](#), [pthread\\_mutex\\_timedlock\(\)](#),  
52802 [pthread\\_mutexattr\\_getrobust\(\)](#)

52803 XBD Section 4.11 (on page 110), [<pthread.h>](#)

**CHANGE HISTORY**

52805 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

52807 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions are  
52808 marked as part of the Threads option.

- 52809 The following new requirements on POSIX implementations derive from alignment with the  
52810 Single UNIX Specification:
- 52811 • The behavior when attempting to relock a mutex is defined.
- 52812 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
52813 IEEE Std 1003.1d-1999.
- 52814 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS  
52815 section so that the [EDEADLK] error includes detection of a deadlock condition. The  
52816 RATIONALE section is also reworded to take into account non-XSI-conformant systems.
- 52817 **Issue 7**
- 52818 SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent  
52819 on the Thread Priority Protection option.
- 52820 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.
- 52821 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions are  
52822 moved from the Threads option to the Base.
- 52823 The following extended mutex types are moved from the XSI option to the Base:
- 52824 PTHREAD\_MUTEX\_NORMAL
  - 52825 PTHREAD\_MUTEX\_ERRORCHECK
  - 52826 PTHREAD\_MUTEX\_RECURSIVE
  - 52827 PTHREAD\_MUTEX\_DEFAULT
- 52828 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized  
52829 mutex.
- 52830 The ERRORS section is updated to account properly for all of the various mutex types.

**pthread\_mutex\_setprioceiling()**

System Interfaces

52831 **NAME**

52832 pthread\_mutex\_setprioceiling — change the priority ceiling of a mutex (**REALTIME**  
52833 **THREADS**)

52834 **SYNOPSIS**

```
52835 RPP|TPP #include <pthread.h>  
52836 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,  
52837 int prioceiling, int *restrict old_ceiling);
```

52838 **DESCRIPTION**

52839 Refer to [pthread\\_mutex\\_getprioceiling\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

52840 **NAME**

52841 pthread\_mutex\_timedlock — lock a mutex

52842 **SYNOPSIS**

52843 #include &lt;pthread.h&gt;

52844 #include &lt;time.h&gt;

52845 int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex,  
52846 const struct timespec \*restrict abstime);52847 **DESCRIPTION**

52848 The *pthread\_mutex\_timedlock()* function shall lock the mutex object referenced by *mutex*. If the  
52849 mutex is already locked, the calling thread shall block until the mutex becomes available as in  
52850 the *pthread\_mutex\_lock()* function. If the mutex cannot be locked without waiting for another  
52851 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

52852 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the  
52853 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
52854 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
52855 call.

52856 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
52857 be the resolution of the clock on which it is based. The **timespec** data type is defined in the  
52858 <time.h> header.

52859 Under no circumstance shall the function fail with a timeout if the mutex can be locked  
52860 immediately. The validity of the *abstime* parameter need not be checked if the mutex can be  
52861 locked immediately.

52862 RPI|TPI As a consequence of the priority inheritance rules (for mutexes initialized with the  
52863 PRIO\_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the  
52864 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this  
52865 thread is no longer among the threads waiting for the mutex.

52866 If *mutex* is a robust mutex and the process containing the owning thread terminated while  
52867 holding the mutex lock, a call to *pthread\_mutex\_timedlock()* shall return the error value  
52868 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding  
52869 the mutex lock, a call to *pthread\_mutex\_timedlock()* may return the error value [EOWNERDEAD]  
52870 even if the process in which the owning thread resides has not terminated. In these cases, the  
52871 mutex is locked by the thread but the state it protects is marked as inconsistent. The application  
52872 should ensure that the state is made consistent for reuse and when that is complete call  
52873 *pthread\_mutex\_consistent()*. If the application is unable to recover the state, it should unlock the  
52874 mutex without a prior call to *pthread\_mutex\_consistent()*, after which the mutex is marked  
52875 permanently unusable.

52876 If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

52877 **RETURN VALUE**

52878 If successful, the *pthread\_mutex\_timedlock()* function shall return zero; otherwise, an error  
52879 number shall be returned to indicate the error.

52880 **ERRORS**

52881 The *pthread\_mutex\_timedlock()* function shall fail if:

52882 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
52883 locks for *mutex* has been exceeded.

**pthread\_mutex\_timedlock()**

System Interfaces

- 52884 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
52885 thread already owns the mutex.
- 52886 [EINVAL] The mutex was created with the protocol attribute having the value  
52887 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
52888 the mutex' current priority ceiling.
- 52889 [EINVAL] The process or thread would have blocked, and the *abstime* parameter  
52890 specified a nanoseconds field value less than zero or greater than or equal to  
52891 1 000 million.
- 52892 [ENOTRECOVERABLE]  
52893 The state protected by the mutex is not recoverable.
- 52894 [EOWNERDEAD]  
52895 The mutex is a robust mutex and the process containing the previous owning  
52896 thread terminated while holding the mutex lock. The mutex lock shall be  
52897 acquired by the calling thread and it is up to the new owner to make the state  
52898 consistent.
- 52899 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.
- 52900 The *pthread\_mutex\_timedlock()* function may fail if:
- 52901 [EDEADLK] A deadlock condition was detected.
- 52902 [EOWNERDEAD]  
52903 The mutex is a robust mutex and the previous owning thread terminated  
52904 while holding the mutex lock. The mutex lock shall be acquired by the calling  
52905 thread and it is up to the new owner to make the state consistent.
- 52906 This function shall not return an error code of [EINTR].

**52907 EXAMPLES**

52908 None.

**52909 APPLICATION USAGE**

52910 Applications that have assumed that non-zero return values are errors will need updating for  
52911 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting  
52912 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error  
52913 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If  
52914 an application is supposed to work with normal and robust mutexes, it should check all return  
52915 values for error conditions and if necessary take appropriate action.

**52916 RATIONALE**52917 Refer to *pthread\_mutex\_lock()*.**52918 FUTURE DIRECTIONS**

52919 None.

**52920 SEE ALSO**52921 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *time()*52922 XBD Section 4.11 (on page 110), *<pthread.h>*, *<time.h>***52923 CHANGE HISTORY**

52924 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52925 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph  
52926 in the DESCRIPTION as part of the Thread Priority Inheritance option.

- 52927 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.
- 52928
- 52929 **Issue 7**
- 52930 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.
- 52931 The *pthread\_mutex\_timedlock()* function is moved from the Timeouts option to the Base.
- 52932 Functionality relating to the Timers option is moved to the Base.
- 52933 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.
- 52934
- 52935 The ERRORS section is updated to account properly for all of the various mutex types.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutex\_trylock()***System Interfaces*52936 **NAME**

52937 pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a mutex

52938 **SYNOPSIS**

52939 #include &lt;pthread.h&gt;

52940 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

52941 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

52942 **DESCRIPTION**52943 Refer to [pthread\\_mutex\\_lock\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

52944 **NAME**

52945 pthread\_mutexattr\_destroy, pthread\_mutexattr\_init — destroy and initialize the mutex  
52946 attributes object

52947 **SYNOPSIS**

```
52948 #include <pthread.h>
52949
52949 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
52950 int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

52951 **DESCRIPTION**

52952 The *pthread\_mutexattr\_destroy()* function shall destroy a mutex attributes object; the object  
52953 becomes, in effect, uninitialized. An implementation may cause *pthread\_mutexattr\_destroy()* to  
52954 set the object referenced by *attr* to an invalid value.

52955 A destroyed *attr* attributes object can be reinitialized using *pthread\_mutexattr\_init()*; the results of  
52956 otherwise referencing the object after it has been destroyed are undefined.

52957 The *pthread\_mutexattr\_init()* function shall initialize a mutex attributes object *attr* with the  
52958 default value for all of the attributes defined by the implementation.

52959 Results are undefined if *pthread\_mutexattr\_init()* is called specifying an already initialized *attr*  
52960 attributes object.

52961 After a mutex attributes object has been used to initialize one or more mutexes, any function  
52962 affecting the attributes object (including destruction) shall not affect any previously initialized  
52963 mutexes.

52964 The behavior is undefined if the value specified by the *attr* argument to  
52965 *pthread\_mutexattr\_destroy()* does not refer to an initialized mutex attributes object.

52966 **RETURN VALUE**

52967 Upon successful completion, *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* shall  
52968 return zero; otherwise, an error number shall be returned to indicate the error.

52969 **ERRORS**

52970 The *pthread\_mutexattr\_init()* function shall fail if:

52971 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

52972 These functions shall not return an error code of [EINTR].

52973 **EXAMPLES**

52974 None.

52975 **APPLICATION USAGE**

52976 None.

52977 **RATIONALE**

52978 If an implementation detects that the value specified by the *attr* argument to  
52979 *pthread\_mutexattr\_destroy()* does not refer to an initialized mutex attributes object, it is  
52980 recommended that the function should fail and report an [EINVAL] error.

52981 See *pthread\_attr\_destroy()* for a general explanation of attributes. Attributes objects allow  
52982 implementations to experiment with useful extensions and permit extension of this volume of  
52983 POSIX.1-2008 without changing the existing functions. Thus, they provide for future  
52984 extensibility of this volume of POSIX.1-2008 and reduce the temptation to standardize  
52985 prematurely on semantics that are not yet widely implemented or understood.

52986 Examples of possible additional mutex attributes that have been discussed are *spin\_only*,  
52987 *limited\_spin*, *no\_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:

**pthread\_mutexattr\_destroy()**

52988 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes  
 52989 would transparently keep records of queue length, wait time, and so on.) Since there is not yet  
 52990 wide agreement on the usefulness of these resulting from shared implementation and usage  
 52991 experience, they are not yet specified in this volume of POSIX.1-2008. Mutex attributes objects,  
 52992 however, make it possible to test out these concepts for possible standardization at a later time.

**Mutex Attributes and Performance**

52993 Care has been taken to ensure that the default values of the mutex attributes have been defined  
 52994 such that mutexes initialized with the defaults have simple enough semantics so that the locking  
 52995 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few  
 52996 other basic instructions).  
 52997

52998 There is at least one implementation method that can be used to reduce the cost of testing at  
 52999 lock-time if a mutex has non-default attributes. One such method that an implementation can  
 53000 employ (and this can be made fully transparent to fully conforming POSIX applications) is to  
 53001 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt  
 53002 to lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were  
 53003 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-  
 53004 default mutex. The underlying unlock operation is more complicated since the implementation  
 53005 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending  
 53006 on the hardware, there may be certain optimizations that can be used so that whatever mutex  
 53007 attributes are considered “most frequently used” can be processed most efficiently.

**Process Shared Memory and Synchronization**

53008 The existence of memory mapping functions in this volume of POSIX.1-2008 leads to the  
 53009 possibility that an application may allocate the synchronization objects from this section in  
 53010 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).  
 53011

53012 In order to permit such usage, while at the same time keeping the usual case (that is, usage  
 53013 within a single process) efficient, a *process-shared* option has been defined.

53014 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the  
 53015 *process-shared* attribute can be used to indicate that mutexes or condition variables may be  
 53016 accessed by threads of multiple processes.

53017 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*  
 53018 attribute so that the most efficient forms of these synchronization objects are created by default.

53019 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*  
 53020 *shared* attribute may only be operated on by threads in the process that initialized them.  
 53021 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*  
 53022 *shared* attribute may be operated on by any thread in any process that has access to it. In  
 53023 particular, these processes may exist beyond the lifetime of the initializing process. For example,  
 53024 the following code implements a simple counting semaphore in a mapped file that may be used  
 53025 by many processes.

```
53026 /* sem.h */
53027 struct semaphore {
53028     pthread_mutex_t lock;
53029     pthread_cond_t nonzero;
53030     unsigned count;
53031 };
53032 typedef struct semaphore semaphore_t;
```

```

53033 semaphore_t *semaphore_create(char *semaphore_name);
53034 semaphore_t *semaphore_open(char *semaphore_name);
53035 void semaphore_post(semaphore_t *semap);
53036 void semaphore_wait(semaphore_t *semap);
53037 void semaphore_close(semaphore_t *semap);

53038 /* sem.c */
53039 #include <sys/types.h>
53040 #include <sys/stat.h>
53041 #include <sys/mman.h>
53042 #include <fcntl.h>
53043 #include <pthread.h>
53044 #include "sem.h"

53045 semaphore_t *
53046 semaphore_create(char *semaphore_name)
53047 {
53048     int fd;
53049     semaphore_t *semap;
53050     pthread_mutexattr_t psharedm;
53051     pthread_condattr_t psharedc;

53052     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
53053     if (fd < 0)
53054         return (NULL);
53055     (void) ftruncate(fd, sizeof(semaphore_t));
53056     (void) pthread_mutexattr_init(&psharedm);
53057     (void) pthread_mutexattr_setpshared(&psharedm,
53058         PTHREAD_PROCESS_SHARED);
53059     (void) pthread_condattr_init(&psharedc);
53060     (void) pthread_condattr_setpshared(&psharedc,
53061         PTHREAD_PROCESS_SHARED);
53062     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
53063         PROT_READ | PROT_WRITE, MAP_SHARED,
53064         fd, 0);
53065     close (fd);
53066     (void) pthread_mutex_init(&semap->lock, &psharedm);
53067     (void) pthread_cond_init(&semap->nonzero, &psharedc);
53068     semap->count = 0;
53069     return (semap);
53070 }

53071 semaphore_t *
53072 semaphore_open(char *semaphore_name)
53073 {
53074     int fd;
53075     semaphore_t *semap;

53076     fd = open(semaphore_name, O_RDWR, 0666);
53077     if (fd < 0)
53078         return (NULL);
53079     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
53080         PROT_READ | PROT_WRITE, MAP_SHARED,
53081         fd, 0);

```

**pthread\_mutexattr\_destroy()**

```

53082         close (fd);
53083         return (semap);
53084     }

53085     void
53086     semaphore_post (semaphore_t *semap)
53087     {
53088         pthread_mutex_lock (&semap->lock);
53089         if (semap->count == 0)
53090             pthread_cond_signal (&semap->nonzero);
53091         semap->count++;
53092         pthread_mutex_unlock (&semap->lock);
53093     }

53094     void
53095     semaphore_wait (semaphore_t *semap)
53096     {
53097         pthread_mutex_lock (&semap->lock);
53098         while (semap->count == 0)
53099             pthread_cond_wait (&semap->nonzero, &semap->lock);
53100         semap->count--;
53101         pthread_mutex_unlock (&semap->lock);
53102     }

53103     void
53104     semaphore_close (semaphore_t *semap)
53105     {
53106         munmap ((void *) semap, sizeof (semaphore_t));
53107     }

```

The following code is for three separate processes that create, post, and wait on a semaphore in the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and decrement the counting semaphore (waiting and waking as required) even though they did not initialize the semaphore.

```

53112     /* create.c */
53113     #include "pthread.h"
53114     #include "sem.h"

53115     int
53116     main ()
53117     {
53118         semaphore_t *semap;

53119         semap = semaphore_create ("/tmp/semaphore");
53120         if (semap == NULL)
53121             exit (1);
53122         semaphore_close (semap);
53123         return (0);
53124     }

53125     /* post */
53126     #include "pthread.h"
53127     #include "sem.h"

53128     int

```

```

53129     main()
53130     {
53131         semaphore_t *semap;
53132
53133         semap = semaphore_open("/tmp/semaphore");
53134         if (semap == NULL)
53135             exit(1);
53136         semaphore_post(semap);
53137         semaphore_close(semap);
53138         return (0);
53139     }
53140
53141     /* wait */
53142     #include "pthread.h"
53143     #include "sem.h"
53144
53145     int
53146     main()
53147     {
53148         semaphore_t *semap;
53149
53150         semap = semaphore_open("/tmp/semaphore");
53151         if (semap == NULL)
53152             exit(1);
53153         semaphore_wait(semap);
53154         semaphore_close(semap);
53155         return (0);
53156     }

```

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

[pthread\\_cond\\_destroy\(\)](#), [pthread\\_create\(\)](#), [pthread\\_mutex\\_destroy\(\)](#)

XBD <pthread.h>

**CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

The [pthread\\_mutexattr\\_destroy\(\)](#) and [pthread\\_mutexattr\\_init\(\)](#) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

**Issue 7**

The [pthread\\_mutexattr\\_destroy\(\)](#) and [pthread\\_mutexattr\\_init\(\)](#) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread\_mutexattr\_getprioceiling()**

System Interfaces

53169 **NAME**

53170 pthread\_mutexattr\_getprioceiling, pthread\_mutexattr\_setprioceiling — get and set the  
 53171 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

53172 **SYNOPSIS**

```
53173 RPP|TPP #include <pthread.h>
53174 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
53175     *restrict attr, int *restrict prioceiling);
53176 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
53177     int prioceiling);
```

53178 **DESCRIPTION**

53179 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions,  
 53180 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to  
 53181 by *attr* which was previously created by the function *pthread\_mutexattr\_init()*.

53182 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of  
 53183 *prioceiling* are within the maximum range of priorities defined by **SCHED\_FIFO**.

53184 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum  
 53185 priority level at which the critical section guarded by the mutex is executed. In order to avoid  
 53186 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal  
 53187 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are  
 53188 within the maximum range of priorities defined under the **SCHED\_FIFO** scheduling policy.

53189 The behavior is undefined if the value specified by the *attr* argument to  
 53190 *pthread\_mutexattr\_getprioceiling()* or *pthread\_mutexattr\_setprioceiling()* does not refer to an  
 53191 initialized mutex attributes object.

53192 **RETURN VALUE**

53193 Upon successful completion, the *pthread\_mutexattr\_getprioceiling()* and  
 53194 *pthread\_mutexattr\_setprioceiling()* functions shall return zero; otherwise, an error number shall be  
 53195 returned to indicate the error.

53196 **ERRORS**

53197 These functions may fail if:

53198 [EINVAL] The value specified by *prioceiling* is invalid.

53199 [EPERM] The caller does not have the privilege to perform the operation.

53200 These functions shall not return an error code of [EINTR].

53201 **EXAMPLES**

53202 None.

53203 **APPLICATION USAGE**

53204 None.

53205 **RATIONALE**

53206 If an implementation detects that the value specified by the *attr* argument to  
 53207 *pthread\_mutexattr\_getprioceiling()* or *pthread\_mutexattr\_setprioceiling()* does not refer to an  
 53208 initialized mutex attributes object, it is recommended that the function should fail and report an  
 53209 [EINVAL] error.

**53210 FUTURE DIRECTIONS**

53211 None.

**53212 SEE ALSO**

53213 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*

53214 XBD <pthread.h>

**53215 CHANGE HISTORY**

53216 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53217 Marked as part of the Realtime Threads Feature Group.

**53218 Issue 6**

53219 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions are marked  
53220 as part of the Threads and Thread Priority Protection options.

53221 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
53222 implementation does not support the Thread Priority Protection option.

53223 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*  
53224 argument.

53225 The **restrict** keyword is added to the *pthread\_mutexattr\_getprioceiling()* prototype for alignment  
53226 with the ISO/IEC 9899:1999 standard.

**53227 Issue 7**

53228 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions are moved  
53229 from the Threads option to require support of either the Robust Mutex Priority Protection option  
53230 or the Non-Robust Mutex Priority Protection option.

53231 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
53232 results in undefined behavior.

**pthread\_mutexattr\_getprotocol()**

System Interfaces

53233 **NAME**

53234 pthread\_mutexattr\_getprotocol, pthread\_mutexattr\_setprotocol — get and set the protocol  
 53235 attribute of the mutex attributes object (**REALTIME THREADS**)

53236 **SYNOPSIS**

```
53237 MC1 #include <pthread.h>
53238
53238 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
53239 *restrict attr, int *restrict protocol);
53240 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
53241 int protocol);
```

53242 **DESCRIPTION**

53243 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions, respectively,  
 53244 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was  
 53245 previously created by the function *pthread\_mutexattr\_init()*.

53246 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of  
 53247 *protocol* may be one of:

53248	RPI   TPI	PTHREAD_PRIO_INHERIT
53249	MC1	PTHREAD_PRIO_NONE
53250	RPP   TPP	PTHREAD_PRIO_PROTECT

53251 which are defined in the **<pthread.h>** header. The default value of the attribute shall be  
 53252 PTHREAD\_PRIO\_NONE.

53253 When a thread owns a mutex with the PTHREAD\_PRIO\_NONE *protocol* attribute, its priority  
 53254 and scheduling shall not be affected by its mutex ownership.

53255 RPI When a thread is blocking higher priority threads because of owning one or more robust  
 53256 mutexes with the PTHREAD\_PRIO\_INHERIT *protocol* attribute, it shall execute at the higher of  
 53257 its priority or the priority of the highest priority thread waiting on any of the robust mutexes  
 53258 owned by this thread and initialized with this protocol.

53259 TPI When a thread is blocking higher priority threads because of owning one or more non-robust  
 53260 mutexes with the PTHREAD\_PRIO\_INHERIT *protocol* attribute, it shall execute at the higher of  
 53261 its priority or the priority of the highest priority thread waiting on any of the non-robust  
 53262 mutexes owned by this thread and initialized with this protocol.

53263 RPP When a thread owns one or more robust mutexes initialized with the  
 53264 PTHREAD\_PRIO\_PROTECT protocol, it shall execute at the higher of its priority or the highest  
 53265 of the priority ceilings of all the robust mutexes owned by this thread and initialized with this  
 53266 attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

53267 TPP When a thread owns one or more non-robust mutexes initialized with the  
 53268 PTHREAD\_PRIO\_PROTECT protocol, it shall execute at the higher of its priority or the highest  
 53269 of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with  
 53270 this attribute, regardless of whether other threads are blocked on any of these non-robust  
 53271 mutexes or not.

53272 While a thread is holding a mutex which has been initialized with the  
 53273 PTHREAD\_PRIO\_INHERIT or PTHREAD\_PRIO\_PROTECT protocol attributes, it shall not be  
 53274 subject to being moved to the tail of the scheduling queue at its priority in the event that its  
 53275 original priority is changed, such as by a call to *sched\_setparam()*. Likewise, when a thread

53276 unlocks a mutex that has been initialized with the PTHREAD\_PRIO\_INHERIT or  
 53277 PTHREAD\_PRIO\_PROTECT protocol attributes, it shall not be subject to being moved to the tail  
 53278 of the scheduling queue at its priority in the event that its original priority is changed.

53279 If a thread simultaneously owns several mutexes initialized with different protocols, it shall  
 53280 execute at the highest of the priorities that it would have obtained by each of these protocols.

53281 RPI | TPI When a thread makes a call to *pthread\_mutex\_lock()*, the mutex was initialized with the protocol  
 53282 attribute having the value PTHREAD\_PRIO\_INHERIT, when the calling thread is blocked  
 53283 because the mutex is owned by another thread, that owner thread shall inherit the priority level  
 53284 of the calling thread as long as it continues to own the mutex. The implementation shall update  
 53285 its execution priority to the maximum of its assigned priority and all its inherited priorities.  
 53286 Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol*  
 53287 attribute having the value PTHREAD\_PRIO\_INHERIT, the same priority inheritance effect shall  
 53288 be propagated to this other owner thread, in a recursive manner.

53289 The behavior is undefined if the value specified by the *attr* argument to  
 53290 *pthread\_mutexattr\_getprotocol()* or *pthread\_mutexattr\_setprotocol()* does not refer to an initialized  
 53291 mutex attributes object.

#### 53292 RETURN VALUE

53293 Upon successful completion, the *pthread\_mutexattr\_getprotocol()* and  
 53294 *pthread\_mutexattr\_setprotocol()* functions shall return zero; otherwise, an error number shall be  
 53295 returned to indicate the error.

#### 53296 ERRORS

53297 The *pthread\_mutexattr\_setprotocol()* function shall fail if:

53298 [ENOTSUP] The value specified by *protocol* is an unsupported value.

53299 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions may fail if:

53300 [EINVAL] The value specified by *protocol* is invalid.

53301 [EPERM] The caller does not have the privilege to perform the operation.

53302 These functions shall not return an error code of [EINTR].

#### 53303 EXAMPLES

53304 None.

#### 53305 APPLICATION USAGE

53306 None.

#### 53307 RATIONALE

53308 If an implementation detects that the value specified by the *attr* argument to  
 53309 *pthread\_mutexattr\_getprotocol()* or *pthread\_mutexattr\_setprotocol()* does not refer to an initialized  
 53310 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]  
 53311 error.

#### 53312 FUTURE DIRECTIONS

53313 None.

#### 53314 SEE ALSO

53315 [pthread\\_cond\\_destroy\(\)](#), [pthread\\_create\(\)](#), [pthread\\_mutex\\_destroy\(\)](#)

53316 XBD [<pthread.h>](#)

**pthread\_mutexattr\_getprotocol()**

System Interfaces

53317 **CHANGE HISTORY**

53318 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53319 Marked as part of the Realtime Threads Feature Group.

53320 **Issue 6**53321 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are marked as  
53322 part of the Threads option and either the Thread Priority Protection or Thread Priority  
53323 Inheritance options.53324 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
53325 implementation does not support the Thread Priority Protection or Thread Priority Inheritance  
53326 options.53327 The **restrict** keyword is added to the *pthread\_mutexattr\_getprotocol()* prototype for alignment  
53328 with the ISO/IEC 9899:1999 standard.53329 **Issue 7**53330 SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the  
53331 *protocol* attribute.

53332 SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.

53333 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are moved from  
53334 the Threads option to require support of either the Non-Robust Mutex Priority Protection option  
53335 or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection  
53336 option or the Robust Mutex Priority Inheritance option.53337 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
53338 results in undefined behavior.

**53339 NAME**

53340 pthread\_mutexattr\_getpshared, pthread\_mutexattr\_setpshared — get and set the process-shared  
53341 attribute

**53342 SYNOPSIS**

```
53343 TSH #include <pthread.h>
53344 int pthread_mutexattr_getpshared(const pthread_mutexattr_t
53345 *restrict attr, int *restrict pshared);
53346 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
53347 int pshared);
```

**53348 DESCRIPTION**

53349 The *pthread\_mutexattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
53350 from the attributes object referenced by *attr*.

53351 The *pthread\_mutexattr\_setpshared()* function shall set the *process-shared* attribute in an initialized  
53352 attributes object referenced by *attr*.

53353 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a mutex to be  
53354 operated upon by any thread that has access to the memory where the mutex is allocated, even if  
53355 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*  
53356 attribute is PTHREAD\_PROCESS\_PRIVATE, the mutex shall only be operated upon by threads  
53357 created within the same process as the thread that initialized the mutex; if threads of differing  
53358 processes attempt to operate on such a mutex, the behavior is undefined. The default value of  
53359 the attribute shall be PTHREAD\_PROCESS\_PRIVATE.

53360 The behavior is undefined if the value specified by the *attr* argument to  
53361 *pthread\_mutexattr\_getpshared()* or *pthread\_mutexattr\_setpshared()* does not refer to an initialized  
53362 mutex attributes object.

**53363 RETURN VALUE**

53364 Upon successful completion, *pthread\_mutexattr\_setpshared()* shall return zero; otherwise, an error  
53365 number shall be returned to indicate the error.

53366 Upon successful completion, *pthread\_mutexattr\_getpshared()* shall return zero and store the value  
53367 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
53368 Otherwise, an error number shall be returned to indicate the error.

**53369 ERRORS**

53370 The *pthread\_mutexattr\_setpshared()* function may fail if:

53371 [EINVAL] The new value specified for the attribute is outside the range of legal values  
53372 for that attribute.

53373 These functions shall not return an error code of [EINTR].

**53374 EXAMPLES**

53375 None.

**53376 APPLICATION USAGE**

53377 None.

**53378 RATIONALE**

53379 If an implementation detects that the value specified by the *attr* argument to  
53380 *pthread\_mutexattr\_getpshared()* or *pthread\_mutexattr\_setpshared()* does not refer to an initialized  
53381 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]  
53382 error.

**pthread\_mutexattr\_getpshared()**53383 **FUTURE DIRECTIONS**

53384 None.

53385 **SEE ALSO**53386 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*53387 XBD [<pthread.h>](#)53388 **CHANGE HISTORY**

53389 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53390 **Issue 6**53391 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked as  
53392 part of the Threads and Thread Process-Shared Synchronization options.53393 The **restrict** keyword is added to the *pthread\_mutexattr\_getpshared()* prototype for alignment  
53394 with the ISO/IEC 9899:1999 standard.53395 **Issue 7**53396 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are moved from  
53397 the Threads option.53398 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
53399 results in undefined behavior.

53400 **NAME**

53401 pthread\_mutexattr\_getrobust, pthread\_mutexattr\_setrobust — get and set the mutex robust  
53402 attribute

53403 **SYNOPSIS**

```
53404 #include <pthread.h>
53405 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
53406     attr, int *restrict robust);
53407 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
53408     int robust);
```

53409 **DESCRIPTION**

53410 The *pthread\_mutexattr\_getrobust()* and *pthread\_mutexattr\_setrobust()* functions, respectively, shall  
53411 get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values  
53412 for *robust* include:

53413 **PTHREAD\_MUTEX\_STALLED**

53414 No special actions are taken if the owner of the mutex is terminated while holding the  
53415 mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.  
53416 This is the default value.

53417 **PTHREAD\_MUTEX\_ROBUST**

53418 If the process containing the owning thread of a robust mutex terminates while holding the  
53419 mutex lock, the next thread that acquires the mutex shall be notified about the termination  
53420 by the return value [EOWNERDEAD] from the locking function. If the owning thread of a  
53421 robust mutex terminates while holding the mutex lock, the next thread that acquires the  
53422 mutex may be notified about the termination by the return value [EOWNERDEAD]. The  
53423 notified thread can then attempt to mark the state protected by the mutex as consistent  
53424 again by a call to *pthread\_mutex\_consistent()*. After a subsequent successful call to  
53425 *pthread\_mutex\_unlock()*, the mutex lock shall be released and can be used normally by other  
53426 threads. If the mutex is unlocked without a call to *pthread\_mutex\_consistent()*, it shall be in a  
53427 permanently unusable state and all attempts to lock the mutex shall fail with the error  
53428 [ENOTRECOVERABLE]. The only permissible operation on such a mutex is  
53429 *pthread\_mutex\_destroy()*.

53430 The behavior is undefined if the value specified by the *attr* argument to  
53431 *pthread\_mutexattr\_getrobust()* or *pthread\_mutexattr\_setrobust()* does not refer to an initialized  
53432 mutex attributes object.

53433 **RETURN VALUE**

53434 Upon successful completion, the *pthread\_mutexattr\_getrobust()* function shall return zero and  
53435 store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter.  
53436 Otherwise, an error value shall be returned to indicate the error. If successful, the  
53437 *pthread\_mutexattr\_setrobust()* function shall return zero; otherwise, an error number shall be  
53438 returned to indicate the error.

53439 **ERRORS**

53440 The *pthread\_mutexattr\_setrobust()* function shall fail if:

53441 [EINVAL] The value of *robust* is invalid.

53442 These functions shall not return an error code of [EINTR].

**pthread\_mutexattr\_getrobust()**

System Interfaces

53443 **EXAMPLES**

53444 None.

53445 **APPLICATION USAGE**

53446 The actions required to make the state protected by the mutex consistent again are solely  
53447 dependent on the application. If it is not possible to make the state of a mutex consistent, robust  
53448 mutexes can be used to notify this situation by calling *pthread\_mutex\_unlock()* without a prior  
53449 call to *pthread\_mutex\_consistent()*.

53450 If the state is declared inconsistent by calling *pthread\_mutex\_unlock()* without a prior call to  
53451 *pthread\_mutex\_consistent()*, a possible approach could be to destroy the mutex and then  
53452 reinitialize it. However, it should be noted that this is possible only in certain situations where  
53453 the state protected by the mutex has to be reinitialized and coordination achieved with other  
53454 threads blocked on the mutex, because otherwise a call to a locking function with a reference to a  
53455 mutex object invalidated by a call to *pthread\_mutex\_destroy()* results in undefined behavior.

53456 **RATIONALE**

53457 If an implementation detects that the value specified by the *attr* argument to  
53458 *pthread\_mutexattr\_getrobust()* or *pthread\_mutexattr\_setrobust()* does not refer to an initialized  
53459 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]  
53460 error.

53461 **FUTURE DIRECTIONS**

53462 None.

53463 **SEE ALSO**53464 *pthread\_mutex\_consistent()*, *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*

53465 XBD &lt;pthread.h&gt;

53466 **CHANGE HISTORY**

53467 First released in Issue 7.

53468 **NAME**

53469 pthread\_mutexattr\_gettype, pthread\_mutexattr\_settype — get and set the mutex type attribute

53470 **SYNOPSIS**

53471 #include &lt;pthread.h&gt;

53472 int pthread\_mutexattr\_gettype(const pthread\_mutexattr\_t \*restrict attr,  
53473 int \*restrict type);

53474 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

53475 **DESCRIPTION**53476 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions, respectively, shall get  
53477 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The  
53478 default value of the *type* attribute is PTHREAD\_MUTEX\_DEFAULT.53479 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types  
53480 include:

## 53481 PTHREAD\_MUTEX\_NORMAL

53482 This type of mutex does not detect deadlock. A thread attempting to relock this mutex  
53483 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a  
53484 different thread results in undefined behavior. Attempting to unlock an unlocked mutex  
53485 results in undefined behavior.

## 53486 PTHREAD\_MUTEX\_ERRORCHECK

53487 This type of mutex provides error checking. A thread attempting to relock this mutex  
53488 without first unlocking it shall return with an error. A thread attempting to unlock a mutex  
53489 which another thread has locked shall return with an error. A thread attempting to unlock  
53490 an unlocked mutex shall return with an error.

## 53491 PTHREAD\_MUTEX\_RECURSIVE

53492 A thread attempting to relock this mutex without first unlocking it shall succeed in locking  
53493 the mutex. The relocking deadlock which can occur with mutexes of type  
53494 PTHREAD\_MUTEX\_NORMAL cannot occur with this type of mutex. Multiple locks of this  
53495 mutex shall require the same number of unlocks to release the mutex before another thread  
53496 can acquire the mutex. A thread attempting to unlock a mutex which another thread has  
53497 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall  
53498 return with an error.

## 53499 PTHREAD\_MUTEX\_DEFAULT

53500 Attempting to recursively lock a mutex of this type results in undefined behavior.  
53501 Attempting to unlock a mutex of this type which was not locked by the calling thread  
53502 results in undefined behavior. Attempting to unlock a mutex of this type which is not  
53503 locked results in undefined behavior. An implementation may map this mutex to one of the  
53504 other mutex types.53505 The behavior is undefined if the value specified by the *attr* argument to  
53506 *pthread\_mutexattr\_gettype()* or *pthread\_mutexattr\_settype()* does not refer to an initialized mutex  
53507 attributes object.53508 **RETURN VALUE**53509 Upon successful completion, the *pthread\_mutexattr\_gettype()* function shall return zero and store  
53510 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,  
53511 an error shall be returned to indicate the error.53512 If successful, the *pthread\_mutexattr\_settype()* function shall return zero; otherwise, an error  
53513 number shall be returned to indicate the error.

**pthread\_mutexattr\_gettype()****53514 ERRORS**

53515 The *pthread\_mutexattr\_settype()* function shall fail if:

53516 [EINVAL] The value *type* is invalid.

53517 These functions shall not return an error code of [EINTR].

**53518 EXAMPLES**

53519 None.

**53520 APPLICATION USAGE**

53521 It is advised that an application should not use a PTHREAD\_MUTEX\_RECURSIVE mutex with  
 53522 condition variables because the implicit unlock performed for a *pthread\_cond\_timedwait()* or  
 53523 *pthread\_cond\_wait()* may not actually release the mutex (if it had been locked multiple times). If  
 53524 this happens, no other thread can satisfy the condition of the predicate.

**53525 RATIONALE**

53526 If an implementation detects that the value specified by the *attr* argument to  
 53527 *pthread\_mutexattr\_gettype()* or *pthread\_mutexattr\_settype()* does not refer to an initialized mutex  
 53528 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

**53529 FUTURE DIRECTIONS**

53530 None.

**53531 SEE ALSO**

53532 [\*pthread\\_cond\\_timedwait\(\)\*](#)

53533 XBD <[pthread.h](#)>

**53534 CHANGE HISTORY**

53535 First released in Issue 5.

**53536 Issue 6**

53537 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for  
 53538 *pthread\_mutexattr\_gettype()* is updated so that the first argument is of type **const**  
 53539 **pthread\_mutexattr\_t\***.

53540 The **restrict** keyword is added to the *pthread\_mutexattr\_gettype()* prototype for alignment with  
 53541 the ISO/IEC 9899:1999 standard.

**53542 Issue 7**

53543 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions are moved from the  
 53544 XSI option to the Base.

53545 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
 53546 results in undefined behavior.

*System Interfaces***pthread\_mutexattr\_init()**53547 **NAME**

53548 pthread\_mutexattr\_init — initialize the mutex attributes object

53549 **SYNOPSIS**

53550 #include &lt;pthread.h&gt;

53551 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

53552 **DESCRIPTION**53553 Refer to *pthread\_mutexattr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutexattr\_setprioceiling()**

System Interfaces

53554 **NAME**

53555 pthread\_mutexattr\_setprioceiling — set the prioceiling attribute of the mutex attributes object  
53556 (**REALTIME THREADS**)

53557 **SYNOPSIS**

```
53558 RPP|TPP #include <pthread.h>  
53559 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
53560 int prioceiling);
```

53561 **DESCRIPTION**

53562 Refer to [pthread\\_mutexattr\\_getprioceiling\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

53563 **NAME**

53564 pthread\_mutexattr\_setprotocol — set the protocol attribute of the mutex attributes object  
53565 (**REALTIME THREADS**)

53566 **SYNOPSIS**

```
53567 MC1 #include <pthread.h>  
53568 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,  
53569 int protocol);
```

53570 **DESCRIPTION**

53571 Refer to [pthread\\_mutexattr\\_getprotocol\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutexattr\_setpshared()***System Interfaces*53572 **NAME**

53573 pthread\_mutexattr\_setpshared — set the process-shared attribute

53574 **SYNOPSIS**

```
53575 TSH #include <pthread.h>
53576      int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
53577      int pshared);
```

53578 **DESCRIPTION**53579 Refer to [pthread\\_mutexattr\\_getpshared\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread\_mutexattr\_setrobust()**53580 **NAME**

53581 pthread\_mutexattr\_setrobust — get and set the mutex robust attribute

53582 **SYNOPSIS**

53583 #include &lt;pthread.h&gt;

53584 int pthread\_mutexattr\_setrobust(pthread\_mutexattr\_t \*attr,  
53585 int robust);53586 **DESCRIPTION**53587 Refer to [pthread\\_mutexattr\\_getrobust\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_mutexattr\_settype()***System Interfaces*53588 **NAME**

53589 pthread\_mutexattr\_settype — set the mutex type attribute

53590 **SYNOPSIS**

53591 #include &lt;pthread.h&gt;

53592 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

53593 **DESCRIPTION**53594 Refer to *pthread\_mutexattr\_gettype()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

53595 **NAME**

53596 pthread\_once — dynamic package initialization

53597 **SYNOPSIS**

```
53598 #include <pthread.h>
53599 int pthread_once(pthread_once_t *once_control,
53600 void (*init_routine)(void));
53601 pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

53602 **DESCRIPTION**

53603 The first call to *pthread\_once()* by any thread in a process, with a given *once\_control*, shall call the  
 53604 *init\_routine* with no arguments. Subsequent calls of *pthread\_once()* with the same *once\_control*  
 53605 shall not call the *init\_routine*. On return from *pthread\_once()*, *init\_routine* shall have completed.  
 53606 The *once\_control* parameter shall determine whether the associated initialization routine has been  
 53607 called.

53608 The *pthread\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation  
 53609 point and is canceled, the effect on *once\_control* shall be as if *pthread\_once()* was never called.

53610 The constant PTHREAD\_ONCE\_INIT is defined in the **<pthread.h>** header.

53611 The behavior of *pthread\_once()* is undefined if *once\_control* has automatic storage duration or is  
 53612 not initialized by PTHREAD\_ONCE\_INIT.

53613 **RETURN VALUE**

53614 Upon successful completion, *pthread\_once()* shall return zero; otherwise, an error number shall  
 53615 be returned to indicate the error.

53616 **ERRORS**

53617 The *pthread\_once()* function shall not return an error code of [EINTR].

53618 **EXAMPLES**

53619 None.

53620 **APPLICATION USAGE**

53621 None.

53622 **RATIONALE**

53623 Some C libraries are designed for dynamic initialization. That is, the global initialization for the  
 53624 library is performed when the first procedure in the library is called. In a single-threaded  
 53625 program, this is normally implemented using a static variable whose value is checked on entry  
 53626 to a routine, as follows:

```
53627 static int random_is_initialized = 0;
53628 extern int initialize_random();
53629 int random_function()
53630 {
53631     if (random_is_initialized == 0) {
53632         initialize_random();
53633         random_is_initialized = 1;
53634     }
53635     ... /* Operations performed after initialization. */
53636 }
```

53637 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,  
 53638 library initialization has to be accomplished by an explicit call to a library-exported initialization  
 53639 function prior to any use of the library.

**pthread\_once()**

53640 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not  
 53641 sufficient; the flag needs to be protected against modification by multiple threads  
 53642 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,  
 53643 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized  
 53644 once requires a recursive solution to this problem.

53645 The use of *pthread\_once()* not only supplies an implementation-guaranteed means of dynamic  
 53646 initialization, it provides an aid to the reliable construction of multi-threaded and realtime  
 53647 systems. The preceding example then becomes:

```
53648 #include <pthread.h>
53649 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
53650 extern int initialize_random();

53651 int random_function()
53652 {
53653     (void) pthread_once(&random_is_initialized, initialize_random);
53654     ... /* Operations performed after initialization. */
53655 }
```

53656 Note that a **pthread\_once\_t** cannot be an array because some compilers do not accept the  
 53657 construct **&<array\_name>**.

53658 If an implementation detects that the value specified by the *once\_control* argument to  
 53659 *pthread\_once()* does not refer to a **pthread\_once\_t** object initialized by PTHREAD\_ONCE\_INIT,  
 53660 it is recommended that the function should fail and report an [EINVAL] error.

**FUTURE DIRECTIONS**

53661 None.

**SEE ALSO**

53664 XBD [<pthread.h>](#)

**CHANGE HISTORY**

53665 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

53668 The *pthread\_once()* function is marked as part of the Threads option.

53669 The [EINVAL] error is added as a “may fail” case for if either argument is invalid.

**Issue 7**

53671 The *pthread\_once()* function is moved from the Threads option to the Base.

53672 The [EINVAL] error for an uninitialized **pthread\_once\_t** object is removed; this condition results  
 53673 in undefined behavior.

53674 **NAME**

53675 pthread\_rwlock\_destroy, pthread\_rwlock\_init — destroy and initialize a read-write lock object

53676 **SYNOPSIS**

```
53677 #include <pthread.h>
53678 int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
53679 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
53680 const pthread_rwlockattr_t *restrict attr);
53681 pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

53682 **DESCRIPTION**

53683 The *pthread\_rwlock\_destroy()* function shall destroy the read-write lock object referenced by  
 53684 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is  
 53685 undefined until the lock is reinitialized by another call to *pthread\_rwlock\_init()*. An  
 53686 implementation may cause *pthread\_rwlock\_destroy()* to set the object referenced by *rwlock* to an  
 53687 invalid value. Results are undefined if *pthread\_rwlock\_destroy()* is called when any thread holds  
 53688 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

53689 The *pthread\_rwlock\_init()* function shall allocate any resources required to use the read-write lock  
 53690 referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced by  
 53691 *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as  
 53692 passing the address of a default read-write lock attributes object. Once initialized, the lock can be  
 53693 used any number of times without being reinitialized. Results are undefined if  
 53694 *pthread\_rwlock\_init()* is called specifying an already initialized read-write lock. Results are  
 53695 undefined if a read-write lock is used without first being initialized.

53696 If the *pthread\_rwlock\_init()* function fails, *rwlock* shall not be initialized and the contents of *rwlock*  
 53697 are undefined.

53698 Only the object referenced by *rwlock* may be used for performing synchronization. The result of  
 53699 referring to copies of that object in calls to *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*,  
 53700 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
 53701 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, or *pthread\_rwlock\_wrlock()* is undefined.

53702 In cases where default read-write lock attributes are appropriate, the macro  
 53703 PTHREAD\_RWLOCK\_INITIALIZER can be used to initialize read-write locks that are statically  
 53704 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread\_rwlock\_init()*  
 53705 with the *attr* parameter specified as NULL, except that no error checks are performed.

53706 The behavior is undefined if the value specified by the *attr* argument to *pthread\_rwlock\_init()*  
 53707 does not refer to an initialized read-write lock attributes object.

53708 **RETURN VALUE**

53709 If successful, the *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions shall return zero;  
 53710 otherwise, an error number shall be returned to indicate the error.

53711 **ERRORS**

53712 The *pthread\_rwlock\_init()* function shall fail if:

53713 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 53714 another read-write lock.

53715 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

53716 [EPERM] The caller does not have the privilege to perform the operation.

53717 These functions shall not return an error code of [EINTR].

**pthread\_rwlock\_destroy()**53718 **EXAMPLES**

53719 None.

53720 **APPLICATION USAGE**53721 Applications using these and related read-write lock functions may be subject to priority  
53722 inversion, as discussed in XBD Section 3.285 (on page 79).53723 **RATIONALE**53724 If an implementation detects that the value specified by the *rwlock* argument to  
53725 *pthread\_rwlock\_destroy()* does not refer to an initialized read-write lock object, it is recommended  
53726 that the function should fail and report an [EINVAL] error.53727 If an implementation detects that the value specified by the *attr* argument to  
53728 *pthread\_rwlock\_init()* does not refer to an initialized read-write lock attributes object, it is  
53729 recommended that the function should fail and report an [EINVAL] error.53730 If an implementation detects that the value specified by the *rwlock* argument to  
53731 *pthread\_rwlock\_destroy()* or *pthread\_rwlock\_init()* refers to a locked read-write lock object, or  
53732 detects that the value specified by the *rwlock* argument to *pthread\_rwlock\_init()* refers to an  
53733 already initialized read-write lock object, it is recommended that the function should fail and  
53734 report an [EBUSY] error.53735 **FUTURE DIRECTIONS**

53736 None.

53737 **SEE ALSO**53738 *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*,  
53739 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*53740 XBD Section 3.285 (on page 79), `<pthread.h>`53741 **CHANGE HISTORY**

53742 First released in Issue 5.

53743 **Issue 6**

53744 The following changes are made for alignment with IEEE Std 1003.1j-2000:

53745 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
53746 now part of the Threads option (previously it was part of the Read-Write Locks option in  
53747 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also  
53748 deleted from the SYNOPSIS.

53749 • The DESCRIPTION is updated as follows:

53750 It explicitly notes allocation of resources upon initialization of a read-write lock  
53751 object.53752 — A paragraph is added specifying that copies of read-write lock objects may not be  
53753 used.53754 • An [EINVAL] error is added to the ERRORS section for *pthread\_rwlock\_init()*, indicating  
53755 that the *rwlock* value is invalid.

53756 • The SEE ALSO section is updated.

53757 The **restrict** keyword is added to the *pthread\_rwlock\_init()* prototype for alignment with the  
53758 ISO/IEC 9899:1999 standard.53759 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION  
53760 USAGE relating to priority inversion.

53761 **Issue 7**

53762 Austin Group Interpretation 1003.1-2001 #048 is applied, adding the  
53763 PTHREAD\_RWLOCK\_INITIALIZER macro.

53764 The *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions are moved from the Threads  
53765 option to the Base.

53766 The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes  
53767 object is removed; this condition results in undefined behavior.

53768 The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock  
53769 object is removed; this condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_rwlock\_rdlock()**

System Interfaces

53770 **NAME**

53771 pthread\_rwlock\_rdlock, pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

53772 **SYNOPSIS**

53773 #include &lt;pthread.h&gt;

53774 int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

53775 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

53776 **DESCRIPTION**53777 The *pthread\_rwlock\_rdlock()* function shall apply a read lock to the read-write lock referenced by  
53778 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are  
53779 no writers blocked on the lock.53780 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are  
53781 executing with the scheduling policies SCHED\_FIFO or SCHED\_RR, the calling thread shall not  
53782 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on  
53783 the lock; otherwise, the calling thread shall acquire the lock.53784 TPS TSP If the Thread Execution Scheduling option is supported, and the threads involved in the lock are  
53785 executing with the SCHED\_SPORADIC scheduling policy, the calling thread shall not acquire  
53786 the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock;  
53787 otherwise, the calling thread shall acquire the lock.53788 If the Thread Execution Scheduling option is not supported, it is implementation-defined  
53789 whether the calling thread acquires the lock when a writer does not hold the lock and there are  
53790 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the  
53791 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the  
53792 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.53793 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the  
53794 *pthread\_rwlock\_rdlock()* function *n* times). If so, the application shall ensure that the thread  
53795 performs matching unlocks (that is, it calls the *pthread\_rwlock\_unlock()* function *n* times).53796 The maximum number of simultaneous read locks that an implementation guarantees can be  
53797 applied to a read-write lock shall be implementation-defined. The *pthread\_rwlock\_rdlock()*  
53798 function may fail if this maximum would be exceeded.53799 The *pthread\_rwlock\_tryrdlock()* function shall apply a read lock as in the *pthread\_rwlock\_rdlock()*  
53800 function, with the exception that the function shall fail if the equivalent *pthread\_rwlock\_rdlock()*  
53801 call would have blocked the calling thread. In no case shall the *pthread\_rwlock\_tryrdlock()*  
53802 function ever block; it always either acquires the lock or fails and returns immediately.

53803 Results are undefined if any of these functions are called with an uninitialized read-write lock.

53804 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the  
53805 signal handler the thread resumes waiting for the read-write lock for reading as if it was not  
53806 interrupted.53807 **RETURN VALUE**53808 If successful, the *pthread\_rwlock\_rdlock()* function shall return zero; otherwise, an error number  
53809 shall be returned to indicate the error.53810 The *pthread\_rwlock\_tryrdlock()* function shall return zero if the lock for reading on the read-write  
53811 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
53812 indicate the error.

**53813 ERRORS**

53814 The *pthread\_rwlock\_tryrdlock()* function shall fail if:

53815 [EBUSY] The read-write lock could not be acquired for reading because a writer holds  
53816 the lock or a writer with the appropriate priority was blocked on it.

53817 The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions may fail if:

53818 [EAGAIN] The read lock could not be acquired because the maximum number of read  
53819 locks for *rwlock* has been exceeded.

53820 The *pthread\_rwlock\_rdlock()* function may fail if:

53821 [EDEADLK] A deadlock condition was detected or the current thread already owns the  
53822 read-write lock for writing.

53823 These functions shall not return an error code of [EINTR].

**53824 EXAMPLES**

53825 None.

**53826 APPLICATION USAGE**

53827 Applications using these functions may be subject to priority inversion, as discussed in XBD  
53828 [Section 3.285](#) (on page 79).

**53829 RATIONALE**

53830 If an implementation detects that the value specified by the *rwlock* argument to  
53831 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_tryrdlock()* does not refer to an initialized read-write  
53832 lock object, it is recommended that the function should fail and report an [EINVAL] error.

**53833 FUTURE DIRECTIONS**

53834 None.

**53835 SEE ALSO**

53836 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*,  
53837 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*

53838 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#)

**53839 CHANGE HISTORY**

53840 First released in Issue 5.

**53841 Issue 6**

53842 The following changes are made for alignment with IEEE Std 1003.1j-2000:

53843 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
53844 now part of the Threads option (previously it was part of the Read-Write Locks option in  
53845 IEEE Std 1003.1j-2000 and also part of the XSI extension).

53846 • The DESCRIPTION is updated as follows:

53847 — Conditions under which writers have precedence over readers are specified.

53848 — Failure of *pthread\_rwlock\_tryrdlock()* is clarified.

53849 — A paragraph on the maximum number of read locks is added.

53850 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and  
53851 [EDEADLK] is deleted as a *pthread\_rwlock\_tryrdlock()* error.

**pthread\_rwlock\_rdlock()**

- 53852           • The SEE ALSO section is updated.
- 53853           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS  
53854           section so that the [EDEADLK] error includes detection of a deadlock condition.
- 53855   **Issue 7**
- 53856           The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions are moved from the Threads  
53857           option to the Base.
- 53858           The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
53859           in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

53860 **NAME**

53861 pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

53862 **SYNOPSIS**

53863 #include &lt;pthread.h&gt;

53864 #include &lt;time.h&gt;

```
53865 int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
53866     const struct timespec *restrict abstime);
```

53867 **DESCRIPTION**

53868 The *pthread\_rwlock\_timedrdlock()* function shall apply a read lock to the read-write lock  
 53869 referenced by *rwlock* as in the *pthread\_rwlock\_rdlock()* function. However, if the lock cannot be  
 53870 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
 53871 when the specified timeout expires. The timeout shall expire when the absolute time specified  
 53872 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value  
 53873 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already  
 53874 been passed at the time of the call.

53875 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
 53876 be the resolution of the CLOCK\_REALTIME clock. The **timespec** data type is defined in the  
 53877 **<time.h>** header. Under no circumstances shall the function fail with a timeout if the lock can be  
 53878 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can  
 53879 be immediately acquired.

53880 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 53881 write lock via a call to *pthread\_rwlock\_timedrdlock()*, upon return from the signal handler the  
 53882 thread shall resume waiting for the lock as if it was not interrupted.

53883 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.  
 53884 The results are undefined if this function is called with an uninitialized read-write lock.

53885 **RETURN VALUE**

53886 The *pthread\_rwlock\_timedrdlock()* function shall return zero if the lock for reading on the read-  
 53887 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 53888 to indicate the error.

53889 **ERRORS**

53890 The *pthread\_rwlock\_timedrdlock()* function shall fail if:

53891 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

53892 The *pthread\_rwlock\_timedrdlock()* function may fail if:

53893 [EAGAIN] The read lock could not be acquired because the maximum number of read  
 53894 locks for lock would be exceeded.

53895 [EDEADLK] A deadlock condition was detected or the calling thread already holds a write  
 53896 lock on *rwlock*.

53897 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000  
 53898 million.

53899 This function shall not return an error code of [EINTR].

**pthread\_rwlock\_timedrdlock()**

System Interfaces

53900 **EXAMPLES**

53901 None.

53902 **APPLICATION USAGE**53903 Applications using this function may be subject to priority inversion, as discussed in XBD  
53904 [Section 3.285](#) (on page 79).53905 **RATIONALE**53906 If an implementation detects that the value specified by the *rwlock* argument to  
53907 *pthread\_rwlock\_timedrdlock()* does not refer to an initialized read-write lock object, it is  
53908 recommended that the function should fail and report an [EINVAL] error.53909 **FUTURE DIRECTIONS**

53910 None.

53911 **SEE ALSO**53912 [pthread\\_rwlock\\_destroy\(\)](#), [pthread\\_rwlock\\_rdlock\(\)](#), [pthread\\_rwlock\\_timedwrlock\(\)](#),  
53913 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)53914 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#), [<time.h>](#)53915 **CHANGE HISTORY**

53916 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53917 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS  
53918 section so that the [EDEADLK] error includes detection of a deadlock condition.53919 **Issue 7**53920 The *pthread\_rwlock\_timedrdlock()* function is moved from the Timeouts option to the Base.53921 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
53922 in undefined behavior.

53923 **NAME**

53924 pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

53925 **SYNOPSIS**

53926 #include &lt;pthread.h&gt;

53927 #include &lt;time.h&gt;

53928 int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict *rwlock*,  
53929 const struct timespec \*restrict *abstime*);53930 **DESCRIPTION**

53931 The *pthread\_rwlock\_timedwrlock()* function shall apply a write lock to the read-write lock  
53932 referenced by *rwlock* as in the *pthread\_rwlock\_wrlock()* function. However, if the lock cannot be  
53933 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
53934 when the specified timeout expires. The timeout shall expire when the absolute time specified  
53935 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value  
53936 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already  
53937 been passed at the time of the call.

53938 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
53939 be the resolution of the CLOCK\_REALTIME clock. The **timespec** data type is defined in the  
53940 **<time.h>** header. Under no circumstances shall the function fail with a timeout if the lock can be  
53941 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can  
53942 be immediately acquired.

53943 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
53944 write lock via a call to *pthread\_rwlock\_timedwrlock()*, upon return from the signal handler the  
53945 thread shall resume waiting for the lock as if it was not interrupted.

53946 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The  
53947 results are undefined if this function is called with an uninitialized read-write lock.

53948 **RETURN VALUE**

53949 The *pthread\_rwlock\_timedwrlock()* function shall return zero if the lock for writing on the read-  
53950 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
53951 to indicate the error.

53952 **ERRORS**53953 The *pthread\_rwlock\_timedwrlock()* function shall fail if:

53954 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

53955 The *pthread\_rwlock\_timedwrlock()* function may fail if:53956 [EDEADLK] A deadlock condition was detected or the calling thread already holds the  
53957 *rwlock*.53958 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000  
53959 million.

53960 This function shall not return an error code of [EINTR].

**pthread\_rwlock\_timedwrlock()**

System Interfaces

53961 **EXAMPLES**

53962 None.

53963 **APPLICATION USAGE**53964 Applications using this function may be subject to priority inversion, as discussed in XBD  
53965 [Section 3.285](#) (on page 79).53966 **RATIONALE**53967 If an implementation detects that the value specified by the *rwlock* argument to  
53968 *pthread\_rwlock\_timedwrlock()* does not refer to an initialized read-write lock object, it is  
53969 recommended that the function should fail and report an [EINVAL] error.53970 **FUTURE DIRECTIONS**

53971 None.

53972 **SEE ALSO**53973 [pthread\\_rwlock\\_destroy\(\)](#), [pthread\\_rwlock\\_rdlock\(\)](#), [pthread\\_rwlock\\_timedrdlock\(\)](#),  
53974 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)53975 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#), [<time.h>](#)53976 **CHANGE HISTORY**

53977 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53978 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS  
53979 section so that the [EDEADLK] error includes detection of a deadlock condition.53980 **Issue 7**53981 The *pthread\_rwlock\_timedwrlock()* function is moved from the Timeouts option to the Base.53982 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
53983 in undefined behavior.

*System Interfaces***pthread\_rwlock\_tryrdlock()**53984 **NAME**

53985 pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

53986 **SYNOPSIS**

53987 #include &lt;pthread.h&gt;

53988 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

53989 **DESCRIPTION**53990 Refer to *pthread\_rwlock\_rdlock()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_rwlock\_trywrlock()**53991 **NAME**

53992 pthread\_rwlock\_trywrlock, pthread\_rwlock\_wrlock — lock a read-write lock object for writing

53993 **SYNOPSIS**

53994 #include &lt;pthread.h&gt;

53995 int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlck);

53996 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlck);

53997 **DESCRIPTION**53998 The *pthread\_rwlock\_trywrlock()* function shall apply a write lock like the *pthread\_rwlock\_wrlock()*  
53999 function, with the exception that the function shall fail if any thread currently holds *rwlck* (for  
54000 reading or writing).54001 The *pthread\_rwlock\_wrlock()* function shall apply a write lock to the read-write lock referenced by  
54002 *rwlck*. The calling thread acquires the write lock if no other thread (reader or writer) holds the  
54003 read-write lock *rwlck*. Otherwise, the thread shall block until it can acquire the lock. The calling  
54004 thread may deadlock if at the time the call is made it holds the read-write lock (whether a read  
54005 or write lock).

54006 Implementations may favor writers over readers to avoid writer starvation.

54007 Results are undefined if any of these functions are called with an uninitialized read-write lock.

54008 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the  
54009 signal handler the thread resumes waiting for the read-write lock for writing as if it was not  
54010 interrupted.54011 **RETURN VALUE**54012 The *pthread\_rwlock\_trywrlock()* function shall return zero if the lock for writing on the read-write  
54013 lock object referenced by *rwlck* is acquired. Otherwise, an error number shall be returned to  
54014 indicate the error.54015 If successful, the *pthread\_rwlock\_wrlock()* function shall return zero; otherwise, an error number  
54016 shall be returned to indicate the error.54017 **ERRORS**54018 The *pthread\_rwlock\_trywrlock()* function shall fail if:54019 [EBUSY] The read-write lock could not be acquired for writing because it was already  
54020 locked for reading or writing.54021 The *pthread\_rwlock\_wrlock()* function may fail if:54022 [EDEADLK] A deadlock condition was detected or the current thread already owns the  
54023 read-write lock for writing or reading.

54024 These functions shall not return an error code of [EINTR].

54025 **EXAMPLES**

54026 None.

54027 **APPLICATION USAGE**54028 Applications using these functions may be subject to priority inversion, as discussed in XBD  
54029 [Section 3.285](#) (on page 79).54030 **RATIONALE**54031 If an implementation detects that the value specified by the *rwlck* argument to  
54032 *pthread\_rwlock\_trywrlock()* or *pthread\_rwlock\_wrlock()* does not refer to an initialized read-write  
54033 lock object, it is recommended that the function should fail and report an [EINVAL] error.

54034 **FUTURE DIRECTIONS**

54035 None.

54036 **SEE ALSO**54037 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,  
54038 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_unlock()*

54039 XBD Section 3.285 (on page 79), Section 4.11 (on page 110), &lt;pthread.h&gt;

54040 **CHANGE HISTORY**

54041 First released in Issue 5.

54042 **Issue 6**

54043 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54044 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
54045 now part of the Threads option (previously it was part of the Read-Write Locks option in  
54046 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54047 • The [EDEADLK] error is deleted as a *pthread\_rwlock\_trywrlock()* error.
- 54048 • The SEE ALSO section is updated.

54049 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS  
54050 section so that the [EDEADLK] error includes detection of a deadlock condition.54051 **Issue 7**54052 The *pthread\_rwlock\_trywrlock()* and *pthread\_rwlock\_wrlock()* functions are moved from the  
54053 Threads option to the Base.54054 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
54055 in undefined behavior.

**pthread\_rwlock\_unlock()**

System Interfaces

54056 **NAME**

54057 pthread\_rwlock\_unlock — unlock a read-write lock object

54058 **SYNOPSIS**

54059 #include &lt;pthread.h&gt;

54060 int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

54061 **DESCRIPTION**

54062 The *pthread\_rwlock\_unlock()* function shall release a lock held on the read-write lock object  
 54063 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the  
 54064 calling thread.

54065 If this function is called to release a read lock from the read-write lock object and there are other  
 54066 read locks currently held on this read-write lock object, the read-write lock object remains in the  
 54067 read locked state. If this function releases the last read lock for this read-write lock object, the  
 54068 read-write lock object shall be put in the unlocked state with no owners.

54069 If this function is called to release a write lock for this read-write lock object, the read-write lock  
 54070 object shall be put in the unlocked state.

54071 If there are threads blocked on the lock when it becomes available, the scheduling policy shall  
 54072 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is  
 54073 supported, when threads executing with the scheduling policies SCHED\_FIFO, SCHED\_RR, or  
 54074 SCHED\_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when  
 54075 the lock becomes available. For equal priority threads, write locks shall take precedence over  
 54076 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-  
 54077 defined whether write locks take precedence over read locks.

54078 Results are undefined if this function is called with an uninitialized read-write lock.

54079 **RETURN VALUE**

54080 If successful, the *pthread\_rwlock\_unlock()* function shall return zero; otherwise, an error number  
 54081 shall be returned to indicate the error.

54082 **ERRORS**

54083 The *pthread\_rwlock\_unlock()* function shall not return an error code of [EINTR].

54084 **EXAMPLES**

54085 None.

54086 **APPLICATION USAGE**

54087 None.

54088 **RATIONALE**

54089 If an implementation detects that the value specified by the *rwlock* argument to  
 54090 *pthread\_rwlock\_unlock()* does not refer to an initialized read-write lock object, it is recommended  
 54091 that the function should fail and report an [EINVAL] error.

54092 If an implementation detects that the value specified by the *rwlock* argument to  
 54093 *pthread\_rwlock\_unlock()* refers to a read-write lock object for which the current thread does not  
 54094 hold a lock, it is recommended that the function should fail and report an [EPERM] error.

54095 **FUTURE DIRECTIONS**

54096 None.

54097 **SEE ALSO**

54098 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,  
 54099 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_trywrlock()*

54100 XBD Section 4.11 (on page 110), **<pthread.h>**

54101 **CHANGE HISTORY**

54102 First released in Issue 5.

54103 **Issue 6**

54104 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54105 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
 54106 now part of the Threads option (previously it was part of the Read-Write Locks option in  
 54107 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54108 • The DESCRIPTION is updated as follows:
  - 54109 — The conditions under which writers have precedence over readers are specified.
  - 54110 — The concept of read-write lock owner is deleted.
- 54111 • The SEE ALSO section is updated.

54112 **Issue 7**

54113 SD5-XSH-ERN-183 is applied.

54114 The *pthread\_rwlock\_unlock()* function is moved from the Threads option to the Base.

54115 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
 54116 in undefined behavior.

54117 The [EPERM] error for a read-write lock object for which the current thread does not hold a lock  
 54118 is removed; this condition results in undefined behavior.

**pthread\_rwlock\_wrlock()***System Interfaces*54119 **NAME**

54120 pthread\_rwlock\_wrlock — lock a read-write lock object for writing

54121 **SYNOPSIS**

54122 #include &lt;pthread.h&gt;

54123 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

54124 **DESCRIPTION**54125 Refer to *pthread\_rwlock\_trywrlock()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**54126 NAME**

54127 `pthread_rwlockattr_destroy`, `pthread_rwlockattr_init` — destroy and initialize the read-write  
54128 lock attributes object

**54129 SYNOPSIS**

```
54130 #include <pthread.h>
54131 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
54132 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

**54133 DESCRIPTION**

54134 The `pthread_rwlockattr_destroy()` function shall destroy a read-write lock attributes object. A  
54135 destroyed `attr` attributes object can be reinitialized using `pthread_rwlockattr_init()`; the results of  
54136 otherwise referencing the object after it has been destroyed are undefined. An implementation  
54137 may cause `pthread_rwlockattr_destroy()` to set the object referenced by `attr` to an invalid value.

54138 The `pthread_rwlockattr_init()` function shall initialize a read-write lock attributes object `attr` with  
54139 the default value for all of the attributes defined by the implementation.

54140 Results are undefined if `pthread_rwlockattr_init()` is called specifying an already initialized `attr`  
54141 attributes object.

54142 After a read-write lock attributes object has been used to initialize one or more read-write locks,  
54143 any function affecting the attributes object (including destruction) shall not affect any previously  
54144 initialized read-write locks.

54145 The behavior is undefined if the value specified by the `attr` argument to  
54146 `pthread_rwlockattr_destroy()` does not refer to an initialized read-write lock attributes object.

**54147 RETURN VALUE**

54148 If successful, the `pthread_rwlockattr_destroy()` and `pthread_rwlockattr_init()` functions shall return  
54149 zero; otherwise, an error number shall be returned to indicate the error.

**54150 ERRORS**

54151 The `pthread_rwlockattr_init()` function shall fail if:

54152 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

54153 These functions shall not return an error code of [EINTR].

**54154 EXAMPLES**

54155 None.

**54156 APPLICATION USAGE**

54157 None.

**54158 RATIONALE**

54159 If an implementation detects that the value specified by the `attr` argument to  
54160 `pthread_rwlockattr_destroy()` does not refer to an initialized read-write lock attributes object, it is  
54161 recommended that the function should fail and report an [EINVAL] error.

**54162 FUTURE DIRECTIONS**

54163 None.

**54164 SEE ALSO**

54165 [\*pthread\\_rwlock\\_destroy\(\)\*](#), [\*pthread\\_rwlockattr\\_getpshared\(\)\*](#)

54166 XBD [\*<pthread.h>\*](#)

**pthread\_rwlockattr\_destroy()**

System Interfaces

54167 **CHANGE HISTORY**

54168 First released in Issue 5.

54169 **Issue 6**

54170 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54171 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
- 54172 now part of the Threads option (previously it was part of the Read-Write Locks option in
- 54173 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54174 • The SEE ALSO section is updated.

54175 **Issue 7**54176 The *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions are moved from the

54177 Threads option to the Base.

54178 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this

54179 condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54180 **NAME**

54181 pthread\_rwlockattr\_getpshared, pthread\_rwlockattr\_setpshared — get and set the process-  
 54182 shared attribute of the read-write lock attributes object

54183 **SYNOPSIS**

```
54184 TSH #include <pthread.h>
54185
54185 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
54186 *restrict attr, int *restrict pshared);
54187 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
54188 int pshared);
```

54189 **DESCRIPTION**

54190 The *pthread\_rwlockattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 54191 from the initialized attributes object referenced by *attr*. The *pthread\_rwlockattr\_setpshared()*  
 54192 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

54193 The *process-shared* attribute shall be set to `PTHREAD_PROCESS_SHARED` to permit a read-write  
 54194 lock to be operated upon by any thread that has access to the memory where the read-write lock  
 54195 is allocated, even if the read-write lock is allocated in memory that is shared by multiple  
 54196 processes. If the *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the read-write lock  
 54197 shall only be operated upon by threads created within the same process as the thread that  
 54198 initialized the read-write lock; if threads of differing processes attempt to operate on such a  
 54199 read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall  
 54200 be `PTHREAD_PROCESS_PRIVATE`.

54201 Additional attributes, their default values, and the names of the associated functions to get and  
 54202 set those attribute values are implementation-defined.

54203 The behavior is undefined if the value specified by the *attr* argument to  
 54204 *pthread\_rwlockattr\_getpshared()* or *pthread\_rwlockattr\_setpshared()* does not refer to an initialized  
 54205 read-write lock attributes object.

54206 **RETURN VALUE**

54207 Upon successful completion, the *pthread\_rwlockattr\_getpshared()* function shall return zero and  
 54208 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*  
 54209 parameter. Otherwise, an error number shall be returned to indicate the error.

54210 If successful, the *pthread\_rwlockattr\_setpshared()* function shall return zero; otherwise, an error  
 54211 number shall be returned to indicate the error.

54212 **ERRORS**

54213 The *pthread\_rwlockattr\_setpshared()* function may fail if:

54214 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 54215 for that attribute.

54216 These functions shall not return an error code of [EINTR].

**pthread\_rwlockattr\_getpshared()**

System Interfaces

54217 **EXAMPLES**

54218 None.

54219 **APPLICATION USAGE**

54220 None.

54221 **RATIONALE**

54222 None.

54223 **FUTURE DIRECTIONS**

54224 None.

54225 **SEE ALSO**54226 *pthread\_rwlock\_destroy()*, *pthread\_rwlockattr\_destroy()*

54227 XBD &lt;pthread.h&gt;

54228 **CHANGE HISTORY**

54229 First released in Issue 5.

54230 **Issue 6**

54231 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54232 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54233
- 54234
- 54235 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 54236 • The SEE ALSO section is updated.

54237 The **restrict** keyword is added to the *pthread\_rwlockattr\_getpshared()* prototype for alignment  
 54238 with the ISO/IEC 9899:1999 standard.

54239 **Issue 7**

54240 The *pthread\_rwlockattr\_getpshared()* and *pthread\_rwlockattr\_setpshared()* functions are moved from  
 54241 the Threads option.

54242 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this  
 54243 condition results in undefined behavior.

54244 **NAME**

54245 pthread\_rwlockattr\_init — initialize the read-write lock attributes object

54246 **SYNOPSIS**

54247 #include &lt;pthread.h&gt;

54248 int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

54249 **DESCRIPTION**54250 Refer to *pthread\_rwlockattr\_destroy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_rwlockattr\_setpshared()**

System Interfaces

54251 **NAME**

54252 pthread\_rwlockattr\_setpshared — set the process-shared attribute of the read-write lock  
54253 attributes object

54254 **SYNOPSIS**

```
54255 TSH #include <pthread.h>  
54256 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
54257 int pshared);
```

54258 **DESCRIPTION**

54259 Refer to [pthread\\_rwlockattr\\_getpshared\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54260 **NAME**

54261 pthread\_self — get the calling thread ID

54262 **SYNOPSIS**

54263 #include &lt;pthread.h&gt;

54264 pthread\_t pthread\_self(void);

54265 **DESCRIPTION**54266 The *pthread\_self()* function shall return the thread ID of the calling thread.54267 **RETURN VALUE**54268 The *pthread\_self()* function shall always be successful and no return value is reserved to indicate  
54269 an error.54270 **ERRORS**

54271 No errors are defined.

54272 **EXAMPLES**

54273 None.

54274 **APPLICATION USAGE**

54275 None.

54276 **RATIONALE**54277 The *pthread\_self()* function provides a capability similar to the *getpid()* function for processes  
54278 and the rationale is the same: the creation call does not provide the thread ID to the created  
54279 thread.54280 **FUTURE DIRECTIONS**

54281 None.

54282 **SEE ALSO**54283 *pthread\_create()*, *pthread\_equal()*

54284 XBD &lt;pthread.h&gt;

54285 **CHANGE HISTORY**

54286 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54287 **Issue 6**54288 The *pthread\_self()* function is marked as part of the Threads option.54289 **Issue 7**

54290 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

54291 The *pthread\_self()* function is moved from the Threads option to the Base.

**pthread\_setcancelstate()**

System Interfaces

54292 **NAME**

54293 pthread\_setcancelstate, pthread\_setcanceltype, pthread\_testcancel — set cancelability state

54294 **SYNOPSIS**

```
54295 #include <pthread.h>
54296 int pthread_setcancelstate(int state, int *oldstate);
54297 int pthread_setcanceltype(int type, int *oldtype);
54298 void pthread_testcancel(void);
```

54299 **DESCRIPTION**

54300 The *pthread\_setcancelstate()* function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD\_CANCEL\_ENABLE and PTHREAD\_CANCEL\_DISABLE.

54304 The *pthread\_setcanceltype()* function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD\_CANCEL\_DEFERRED and PTHREAD\_CANCEL\_ASYNCHRONOUS.

54308 The cancelability state and type of any newly created threads, including the thread in which *main()* was first invoked, shall be PTHREAD\_CANCEL\_ENABLE and PTHREAD\_CANCEL\_DEFERRED respectively.

54311 The *pthread\_testcancel()* function shall create a cancellation point in the calling thread. The *pthread\_testcancel()* function shall have no effect if cancelability is disabled.

54313 **RETURN VALUE**

54314 If successful, the *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

54316 **ERRORS**

54317 The *pthread\_setcancelstate()* function may fail if:

54318 [EINVAL] The specified state is not PTHREAD\_CANCEL\_ENABLE or PTHREAD\_CANCEL\_DISABLE.

54320 The *pthread\_setcanceltype()* function may fail if:

54321 [EINVAL] The specified type is not PTHREAD\_CANCEL\_DEFERRED or PTHREAD\_CANCEL\_ASYNCHRONOUS.

54323 These functions shall not return an error code of [EINTR].

54324 **EXAMPLES**

54325 None.

54326 **APPLICATION USAGE**

54327 None.

54328 **RATIONALE**

54329 The *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions control the points at which a thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, some rules need to be followed.

54332 An object can be considered to be a generalization of a procedure. It is a set of procedures and global variables written as a unit and called by clients not known by the object. Objects may depend on other objects.

54335 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On

54336 exit from an object, the cancelability state should always be restored to its value on entry to the  
54337 object.

54338 This follows from a modularity argument: if the client of an object (or the client of an object that  
54339 uses that object) has disabled cancelability, it is because the client does not want to be concerned  
54340 about cleaning up if the thread is canceled while executing some sequence of actions. If an object  
54341 is called in such a state and it enables cancelability and a cancellation request is pending for that  
54342 thread, then the thread is canceled, contrary to the wish of the client that disabled.

54343 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry  
54344 to an object. But as with the cancelability state, on exit from an object the cancelability type  
54345 should always be restored to its value on entry to the object.

54346 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously  
54347 cancelable.

#### 54348 **FUTURE DIRECTIONS**

54349 None.

#### 54350 **SEE ALSO**

54351 [\*pthread\\_cancel\(\)\*](#)

54352 XBD [`<pthread.h>`](#)

#### 54353 **CHANGE HISTORY**

54354 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 54355 **Issue 6**

54356 The [\*pthread\\_setcancelstate\(\)\*](#), [\*pthread\\_setcanceltype\(\)\*](#), and [\*pthread\\_testcancel\(\)\*](#) functions are marked  
54357 as part of the Threads option.

#### 54358 **Issue 7**

54359 The [\*pthread\\_setcancelstate\(\)\*](#), [\*pthread\\_setcanceltype\(\)\*](#), and [\*pthread\\_testcancel\(\)\*](#) functions are moved  
54360 from the Threads option to the Base.

**pthread\_setconcurrency()***System Interfaces*54361 **NAME**

54362 pthread\_setconcurrency — set the level of concurrency

54363 **SYNOPSIS**

54364 OB XSI #include &lt;pthread.h&gt;

54365 int pthread\_setconcurrency(int new\_level);

54366 **DESCRIPTION**54367 Refer to *pthread\_getconcurrency()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54368 **NAME**

54369 pthread\_setschedparam — dynamic thread scheduling parameters access (REALTIME  
54370 THREADS)

54371 **SYNOPSIS**

```
54372 TPS #include <pthread.h>  
54373 int pthread_setschedparam(pthread_t thread, int policy,  
54374 const struct sched_param *param);
```

54375 **DESCRIPTION**

54376 Refer to [pthread\\_getschedparam\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_setschedprio()**

System Interfaces

54377 **NAME**

54378 pthread\_setschedprio — dynamic thread scheduling parameters access (**REALTIME**  
54379 **THREADS**)

54380 **SYNOPSIS**

```
54381 TPS #include <pthread.h>
54382 int pthread_setschedprio(pthread_t thread, int prio);
```

54383 **DESCRIPTION**

54384 The *pthread\_setschedprio()* function shall set the scheduling priority for the thread whose thread  
54385 ID is given by *thread* to the value given by *prio*. See [Scheduling Policies](#) (on page 501) for a  
54386 description on how this function call affects the ordering of the thread in the thread list for its  
54387 new priority.

54388 If the *pthread\_setschedprio()* function fails, the scheduling priority of the target thread shall not be  
54389 changed.

54390 **RETURN VALUE**

54391 If successful, the *pthread\_setschedprio()* function shall return zero; otherwise, an error number  
54392 shall be returned to indicate the error.

54393 **ERRORS**

54394 The *pthread\_setschedprio()* function may fail if:

54395 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

54396 [ENOTSUP] An attempt was made to set the priority to an unsupported value.

54397 [EPERM] The caller does not have appropriate privileges to set the scheduling priority  
54398 of the specified thread.

54399 The *pthread\_setschedprio()* function shall not return an error code of [EINTR].

54400 **EXAMPLES**

54401 None.

54402 **APPLICATION USAGE**

54403 None.

54404 **RATIONALE**

54405 The *pthread\_setschedprio()* function provides a way for an application to temporarily raise its  
54406 priority and then lower it again, without having the undesired side-effect of yielding to other  
54407 threads of the same priority. This is necessary if the application is to implement its own  
54408 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This  
54409 capability is especially important if the implementation does not support the Thread Priority  
54410 Protection or Thread Priority Inheritance options, but even if those options are supported it is  
54411 needed if the application is to bound priority inheritance for other resources, such as  
54412 semaphores.

54413 The standard developers considered that while it might be preferable conceptually to solve this  
54414 problem by modifying the specification of *pthread\_setschedparam()*, it was too late to make such a  
54415 change, as there may be implementations that would need to be changed. Therefore, this new  
54416 function was introduced.

54417 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
54418 that the function should fail and report an [ESRCH] error.

54419 **FUTURE DIRECTIONS**

54420 None.

54421 **SEE ALSO**54422 [Scheduling Policies](#) (on page 501), [pthread\\_getschedparam\(\)](#)54423 XBD [<pthread.h>](#)54424 **CHANGE HISTORY**

54425 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

54426 **Issue 7**54427 The `pthread_setschedprio()` function is moved from the Threads option.

54428 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPEM] error.

54429 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pthread\_setspecific()***System Interfaces*54430 **NAME**

54431 pthread\_setspecific — thread-specific data management

54432 **SYNOPSIS**

54433 #include &lt;pthread.h&gt;

54434 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

54435 **DESCRIPTION**54436 Refer to *pthread\_getspecific()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54437 **NAME**

54438 pthread\_sigmask, sigprocmask — examine and change blocked signals

54439 **SYNOPSIS**

```
54440 CX #include <signal.h>
54441 int pthread_sigmask(int how, const sigset_t *restrict set,
54442 sigset_t *restrict oset);
54443 int sigprocmask(int how, const sigset_t *restrict set,
54444 sigset_t *restrict oset);
```

54445 **DESCRIPTION**

54446 The *pthread\_sigmask()* function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to *sigprocmask()*, without the restriction that the call be made in a single-threaded process.

54449 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the signal mask of the calling thread.

54451 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

54453 The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

54455 SIG\_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.

54457 SIG\_SETMASK The resulting set shall be the signal set pointed to by *set*.

54458 SIG\_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.

54460 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the thread's signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

54464 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those signals shall be delivered before the call to *sigprocmask()* returns.

54466 It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

54468 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()* function, or the *raise()* function.

54471 If *sigprocmask()* fails, the thread's signal mask shall not be changed.

54472 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.

54473 **RETURN VALUE**

54474 Upon successful completion *pthread\_sigmask()* shall return 0; otherwise, it shall return the corresponding error number.

54476 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and the signal mask of the process shall be unchanged.

**pthread\_sigmask()**54478 **ERRORS**54479 The *pthread\_sigmask()* and *sigprocmask()* functions shall fail if:54480 [EINVAL] The value of the *how* argument is not equal to one of the defined values.54481 The *pthread\_sigmask()* function shall not return an error code of [EINTR].54482 **EXAMPLES**54483 **Signaling in a Multi-Threaded Process**54484 This example shows the use of *pthread\_sigmask()* in order to deal with signals in a multi-  
54485 threaded process. It provides a fairly general framework that could be easily adapted/extended.

```

54486 #include <stdio.h>
54487 #include <stdlib.h>
54488 #include <pthread.h>
54489 #include <signal.h>
54490 #include <string.h>
54491 #include <errno.h>
54492 ...
54493 static sigset_t  signal_mask; /* signals to block          */
54494 int main (int argc, char *argv[])
54495 {
54496     pthread_t  sig_thr_id; /* signal handler thread ID */
54497     int        rc; /* return code          */
54498     sigemptyset (&signal_mask);
54499     sigaddset (&signal_mask, SIGINT);
54500     sigaddset (&signal_mask, SIGTERM);
54501     rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
54502     if (rc != 0) {
54503         /* handle error */
54504         ...
54505     }
54506     /* any newly created threads inherit the signal mask */
54507     rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
54508     if (rc != 0) {
54509         /* handle error */
54510         ...
54511     }
54512     /* APPLICATION CODE */
54513     ...
54514 }
54515 void *signal_thread (void *arg)
54516 {
54517     int        sig_caught; /* signal caught          */
54518     int        rc; /* returned code          */
54519     rc = sigwait (&signal_mask, &sig_caught);
54520     if (rc != 0) {
54521         /* handle error */
54522     }

```

```

54523         switch (sig_caught)
54524         {
54525             case SIGINT:      /* process SIGINT */
54526                 ...
54527                 break;
54528             case SIGTERM:    /* process SIGTERM */
54529                 ...
54530                 break;
54531             default:         /* should normally not happen */
54532                 fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
54533                 break;
54534         }
54535     }

```

**54536 APPLICATION USAGE**

54537 None.

**54538 RATIONALE**

54539 When a thread's signal mask is changed in a signal-catching function that is installed by  
54540 *sigaction()*, the restoration of the signal mask on return from the signal-catching function  
54541 overrides that change (see *sigaction()*). If the signal-catching function was installed with  
54542 *signal()*, it is unspecified whether this occurs.

54543 See *kill()* for a discussion of the requirement on delivery of signals.

**54544 FUTURE DIRECTIONS**

54545 None.

**54546 SEE ALSO**

54547 *exec*, *kill()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
54548 *sigpending()*, *sigqueue()*, *sigsuspend()*

54549 XBD <[signal.h](#)>

**54550 CHANGE HISTORY**

54551 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**54552 Issue 5**

54553 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

54554 The *pthread\_sigmask()* function is added for alignment with the POSIX Threads Extension.

**54555 Issue 6**

54556 The *pthread\_sigmask()* function is marked as part of the Threads option.

54557 The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this  
54558 function in the <[signal.h](#)> header is an extension to the ISO C standard.

54559 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 54560 • The DESCRIPTION is updated to explicitly state the functions which may generate the  
54561 signal.

54562 The normative text is updated to avoid use of the term "must" for application requirements.

54563 The **restrict** keyword is added to the *pthread\_sigmask()* and *sigprocmask()* prototypes for  
54564 alignment with the ISO/IEC 9899: 1999 standard.

54565 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating "process' signal  
54566 mask" to "thread's signal mask" in the DESCRIPTION and RATIONALE sections.

**pthread\_sigmask()**

- 54567 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the  
54568 EXAMPLES section.
- 54569 **Issue 7**  
54570 The *pthread\_sigmask()* function is moved from the Threads option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54571 **NAME**

54572 pthread\_spin\_destroy, pthread\_spin\_init — destroy or initialize a spin lock object

54573 **SYNOPSIS**

54574 #include &lt;pthread.h&gt;

54575 int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

54576 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

54577 **DESCRIPTION**

54578 The *pthread\_spin\_destroy()* function shall destroy the spin lock referenced by *lock* and release any  
 54579 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is  
 54580 reinitialized by another call to *pthread\_spin\_init()*. The results are undefined if  
 54581 *pthread\_spin\_destroy()* is called when a thread holds the lock, or if this function is called with an  
 54582 uninitialized thread spin lock.

54583 The *pthread\_spin\_init()* function shall allocate any resources required to use the spin lock  
 54584 referenced by *lock* and initialize the lock to an unlocked state.

54585 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 54586 PTHREAD\_PROCESS\_SHARED, the implementation shall permit the spin lock to be operated  
 54587 upon by any thread that has access to the memory where the spin lock is allocated, even if it is  
 54588 allocated in memory that is shared by multiple processes.

54589 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 54590 PTHREAD\_PROCESS\_PRIVATE, or if the option is not supported, the spin lock shall only be  
 54591 operated upon by threads created within the same process as the thread that initialized the spin  
 54592 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is  
 54593 undefined.

54594 The results are undefined if *pthread\_spin\_init()* is called specifying an already initialized spin  
 54595 lock. The results are undefined if a spin lock is used without first being initialized.

54596 If the *pthread\_spin\_init()* function fails, the lock is not initialized and the contents of *lock* are  
 54597 undefined.

54598 Only the object referenced by *lock* may be used for performing synchronization.

54599 The result of referring to copies of that object in calls to *pthread\_spin\_destroy()*,  
 54600 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, or *pthread\_spin\_unlock()* is undefined.

54601 **RETURN VALUE**

54602 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 54603 be returned to indicate the error.

54604 **ERRORS**

54605 The *pthread\_spin\_init()* function shall fail if:

54606 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

54607 [ENOMEM] Insufficient memory exists to initialize the lock.

54608 These functions shall not return an error code of [EINTR].

**pthread\_spin\_destroy()**

System Interfaces

54609 **EXAMPLES**

54610 None.

54611 **APPLICATION USAGE**

54612 None.

54613 **RATIONALE**

54614 If an implementation detects that the value specified by the *lock* argument to  
 54615 *pthread\_spin\_destroy()* does not refer to an initialized spin lock object, it is recommended that the  
 54616 function should fail and report an [EINVAL] error.

54617 If an implementation detects that the value specified by the *lock* argument to  
 54618 *pthread\_spin\_destroy()* or *pthread\_spin\_init()* refers to a locked spin lock object, or detects that the  
 54619 value specified by the *lock* argument to *pthread\_spin\_init()* refers to an already initialized spin  
 54620 lock object, it is recommended that the function should fail and report an [EBUSY] error.

54621 **FUTURE DIRECTIONS**

54622 None.

54623 **SEE ALSO**54624 *pthread\_spin\_lock()*, *pthread\_spin\_unlock()*54625 XBD `<pthread.h>`54626 **CHANGE HISTORY**

54627 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54628 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.54629 **Issue 7**

54630 The *pthread\_spin\_destroy()* and *pthread\_spin\_init()* functions are moved from the Spin Locks  
 54631 option to the Base.

54632 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
 54633 undefined behavior.

54634 The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is  
 54635 removed; this condition results in undefined behavior.

54636 **NAME**

54637 pthread\_spin\_lock, pthread\_spin\_trylock — lock a spin lock object

54638 **SYNOPSIS**

54639 #include &lt;pthread.h&gt;

54640 int pthread\_spin\_lock(pthread\_spinlock\_t \*lock);

54641 int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

54642 **DESCRIPTION**

54643 The *pthread\_spin\_lock()* function shall lock the spin lock referenced by *lock*. The calling thread shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, shall not return from the *pthread\_spin\_lock()* call) until the lock becomes available. The results are undefined if the calling thread holds the lock at the time the call is made. The *pthread\_spin\_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function shall fail.

54649 The results are undefined if any of these functions is called with an uninitialized spin lock.

54650 **RETURN VALUE**

54651 Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

54653 **ERRORS**54654 The *pthread\_spin\_lock()* function may fail if:

54655 [EDEADLK] A deadlock condition was detected.

54656 The *pthread\_spin\_trylock()* function shall fail if:

54657 [EBUSY] A thread currently holds the lock.

54658 These functions shall not return an error code of [EINTR].

54659 **EXAMPLES**

54660 None.

54661 **APPLICATION USAGE**

54662 Applications using this function may be subject to priority inversion, as discussed in XBD  
54663 Section 3.285 (on page 79).

54664 **RATIONALE**

54665 If an implementation detects that the value specified by the *lock* argument to *pthread\_spin\_lock()*  
54666 or *pthread\_spin\_trylock()* does not refer to an initialized spin lock object, it is recommended that  
54667 the function should fail and report an [EINVAL] error.

54668 If an implementation detects that the value specified by the *lock* argument to *pthread\_spin\_lock()*  
54669 refers to a spin lock object for which the calling thread already holds the lock, it is recommended  
54670 that the function should fail and report an [EDEADLK] error.

54671 **FUTURE DIRECTIONS**

54672 None.

54673 **SEE ALSO**54674 *pthread\_spin\_destroy()*, *pthread\_spin\_unlock()*

54675 XBD Section 3.285 (on page 79), Section 4.11 (on page 110), &lt;pthread.h&gt;

**pthread\_spin\_lock()**

System Interfaces

54676 **CHANGE HISTORY**

54677 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54678 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.54679 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS  
54680 section so that the [EDEADLK] error includes detection of a deadlock condition.54681 **Issue 7**54682 The `pthread_spin_lock()` and `pthread_spin_trylock()` functions are moved from the Spin Locks  
54683 option to the Base.54684 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
54685 undefined behavior.54686 The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is  
54687 removed; this condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54688 **NAME**

54689 pthread\_spin\_unlock — unlock a spin lock object

54690 **SYNOPSIS**

54691 #include &lt;pthread.h&gt;

54692 int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

54693 **DESCRIPTION**54694 The *pthread\_spin\_unlock()* function shall release the spin lock referenced by *lock* which was  
54695 locked via the *pthread\_spin\_lock()* or *pthread\_spin\_trylock()* functions.

54696 The results are undefined if the lock is not held by the calling thread.

54697 If there are threads spinning on the lock when *pthread\_spin\_unlock()* is called, the lock becomes  
54698 available and an unspecified spinning thread shall acquire the lock.

54699 The results are undefined if this function is called with an uninitialized thread spin lock.

54700 **RETURN VALUE**54701 Upon successful completion, the *pthread\_spin\_unlock()* function shall return zero; otherwise, an  
54702 error number shall be returned to indicate the error.54703 **ERRORS**

54704 This function shall not return an error code of [EINTR].

54705 **EXAMPLES**

54706 None.

54707 **APPLICATION USAGE**

54708 None.

54709 **RATIONALE**54710 If an implementation detects that the value specified by the *lock* argument to  
54711 *pthread\_spin\_unlock()* does not refer to an initialized spin lock object, it is recommended that the  
54712 function should fail and report an [EINVAL] error.54713 If an implementation detects that the value specified by the *lock* argument to  
54714 *pthread\_spin\_unlock()* refers to a spin lock object for which the current thread does not hold the  
54715 lock, it is recommended that the function should fail and report an [EPERM] error.54716 **FUTURE DIRECTIONS**

54717 None.

54718 **SEE ALSO**54719 *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*

54720 XBD Section 4.11 (on page 110), &lt;pthread.h&gt;

54721 **CHANGE HISTORY**

54722 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54723 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

54724 **Issue 7**54725 The *pthread\_spin\_unlock()* function is moved from the Spin Locks option to the Base.54726 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
54727 undefined behavior.

## **pthread\_spin\_unlock()**

*System Interfaces*

54728  
54729

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread\_testcancel()**54730 **NAME**

54731 pthread\_testcancel — set cancelability state

54732 **SYNOPSIS**

54733 #include &lt;pthread.h&gt;

54734 void pthread\_testcancel(void);

54735 **DESCRIPTION**54736 Refer to *pthread\_setcancelstate()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ptsname()**54737 **NAME**

54738 ptsname — get name of the slave pseudo-terminal device

54739 **SYNOPSIS**

```
54740 xSI #include <stdlib.h>
54741 char *ptsname(int fildes);
```

54742 **DESCRIPTION**

54743 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated  
 54744 with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the  
 54745 master device. The *ptsname()* function shall return a pointer to a string containing the pathname  
 54746 of the corresponding slave device.

54747 The *ptsname()* function need not be thread-safe.54748 **RETURN VALUE**

54749 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the  
 54750 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could  
 54751 occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file  
 54752 system.

54753 **ERRORS**

54754 No errors are defined.

54755 **EXAMPLES**

54756 None.

54757 **APPLICATION USAGE**54758 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.54759 **RATIONALE**

54760 None.

54761 **FUTURE DIRECTIONS**

54762 None.

54763 **SEE ALSO**54764 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*

54765 XBD &lt;stdlib.h&gt;

54766 **CHANGE HISTORY**

54767 First released in Issue 4, Version 2.

54768 **Issue 5**

54769 Moved from X/OPEN UNIX extension to BASE.

54770 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

54771 **Issue 7**

54772 Austin Group Interpretation 1003.1-2001 #156 is applied.

54773 **NAME**54774        `putc` — put a byte on a stream54775 **SYNOPSIS**54776        `#include <stdio.h>`54777        `int putc(int c, FILE *stream);`54778 **DESCRIPTION**54779 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
54780 conflict between the requirements described here and the ISO C standard is unintentional. This  
54781 volume of POSIX.1-2008 defers to the ISO C standard.54782        The `putc()` function shall be equivalent to `fputc()`, except that if it is implemented as a macro it  
54783 may evaluate `stream` more than once, so the argument should never be an expression with side-  
54784 effects.54785 **RETURN VALUE**54786        Refer to `fputc()`.54787 **ERRORS**54788        Refer to `fputc()`.54789 **EXAMPLES**

54790        None.

54791 **APPLICATION USAGE**54792        Since it may be implemented as a macro, `putc()` may treat a `stream` argument with side-effects  
54793 incorrectly. In particular, `putc(c,*f++)` does not necessarily work correctly. Therefore, use of this  
54794 function is not recommended in such situations; `fputc()` should be used instead.54795 **RATIONALE**

54796        None.

54797 **FUTURE DIRECTIONS**

54798        None.

54799 **SEE ALSO**54800        `fputc()`54801        XBD `<stdio.h>`54802 **CHANGE HISTORY**

54803        First released in Issue 1. Derived from Issue 1 of the SVID.

**putc\_unlocked()***System Interfaces*54804 **NAME**

54805         putc\_unlocked — stdio with explicit client locking

54806 **SYNOPSIS**54807 CX         #include <stdio.h>  
54808         int putc\_unlocked(int c, FILE \*stream);54809 **DESCRIPTION**54810         Refer to *getc\_unlocked()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54811 **NAME**54812            **putchar** — put a byte on a stdout stream54813 **SYNOPSIS**

54814            #include &lt;stdio.h&gt;

54815            int putchar(int c);

54816 **DESCRIPTION**

54817 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 54818 conflict between the requirements described here and the ISO C standard is unintentional. This  
 54819 volume of POSIX.1-2008 defers to the ISO C standard.

54820            The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.54821 **RETURN VALUE**54822            Refer to *fputc()*.54823 **ERRORS**54824            Refer to *fputc()*.54825 **EXAMPLES**

54826            None.

54827 **APPLICATION USAGE**

54828            None.

54829 **RATIONALE**

54830            None.

54831 **FUTURE DIRECTIONS**

54832            None.

54833 **SEE ALSO**54834            *putc()*

54835            XBD &lt;stdio.h&gt;

54836 **CHANGE HISTORY**

54837            First released in Issue 1. Derived from Issue 1 of the SVID.

**putchar\_unlocked()***System Interfaces*54838 **NAME**

54839 putchar\_unlocked — stdio with explicit client locking

54840 **SYNOPSIS**

```
54841 CX #include <stdio.h>  
54842 int putchar_unlocked(int c);
```

54843 **DESCRIPTION**54844 Refer to *getc\_unlocked()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54845 **NAME**

54846 putenv — change or add a value to an environment

54847 **SYNOPSIS**

```
54848 XSI #include <stdlib.h>
54849 int putenv(char *string);
```

54850 **DESCRIPTION**

54851 The *putenv()* function shall use the *string* argument to set environment variable values. The  
 54852 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall  
 54853 make the value of the environment variable *name* equal to *value* by altering an existing variable  
 54854 or creating a new one. In either case, the string pointed to by *string* shall become part of the  
 54855 environment, so altering the string shall change the environment. The space used by *string* is no  
 54856 longer used once a new string which defines *name* is passed to *putenv()*.

54857 The *putenv()* function need not be thread-safe.

54858 **RETURN VALUE**

54859 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value  
 54860 and set *errno* to indicate the error.

54861 **ERRORS**

54862 The *putenv()* function may fail if:

54863 [ENOMEM] Insufficient memory was available.

54864 **EXAMPLES**54865 **Changing the Value of an Environment Variable**

54866 The following example changes the value of the *HOME* environment variable to the value  
 54867 */usr/home*.

```
54868 #include <stdlib.h>
54869 ...
54870 static char *var = "HOME=/usr/home";
54871 int ret;
54872 ret = putenv(var);
```

54873 **APPLICATION USAGE**

54874 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in  
 54875 conjunction with *getenv()*.

54876 See *exec()* for restrictions on changing the environment in multi-threaded applications.

54877 This routine may use *malloc()* to enlarge the environment.

54878 A potential error is to call *putenv()* with an automatic variable as the argument, then return from  
 54879 the calling function while *string* is still part of the environment.

54880 The *setenv()* function is preferred over this function.

54881 **RATIONALE**

54882 The standard developers noted that *putenv()* is the only function available to add to the  
 54883 environment without permitting memory leaks.

**putenv()**54884 **FUTURE DIRECTIONS**

54885 None.

54886 **SEE ALSO**54887 *exec*, *getenv()*, *malloc()*, *setenv()*54888 XBD `<stdlib.h>`54889 **CHANGE HISTORY**

54890 First released in Issue 1. Derived from Issue 1 of the SVID.

54891 **Issue 5**54892 The type of the argument to this function is changed from **const char \*** to **char \***. This was  
54893 indicated as a FUTURE DIRECTION in previous issues.

54894 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

54895 **Issue 6**54896 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the  
54897 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to  
54898 *exec*.54899 **Issue 7**

54900 Austin Group Interpretation 1003.1-2001 #156 is applied.

54901 **NAME**

54902 putmsg, putpmsg — send a message on a STREAM (STREAMS)

54903 **SYNOPSIS**

```
54904 OB XSR #include <stropts.h>
54905 int putmsg(int fildes, const struct strbuf *ctlptr,
54906           const struct strbuf *dataptr, int flags);
54907 int putpmsg(int fildes, const struct strbuf *ctlptr,
54908            const struct strbuf *dataptr, int band, int flags);
```

54909 **DESCRIPTION**

54910 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a  
 54911 STREAMS file. The message may contain either a data part, a control part, or both. The data and  
 54912 control parts are distinguished by placement in separate buffers, as described below. The  
 54913 semantics of each part are defined by the STREAMS module that receives the message.

54914 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in  
 54915 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to  
 54916 *putpmsg()*.

54917 The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and  
 54918 *dataptr* arguments each point to a **strbuf** structure.

54919 The *ctlptr* argument points to the structure describing the control part, if any, to be included in  
 54920 the message. The *buf* member in the **strbuf** structure points to the buffer where the control  
 54921 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*  
 54922 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if  
 54923 any, to be included in the message. The *flags* argument indicates what type of message should be  
 54924 sent and is described further below.

54925 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer  
 54926 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the  
 54927 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part  
 54928 shall be sent if either *dataptr* (*ctlptr*) is a null pointer or the *len* member of *dataptr* (*ctlptr*) is set to  
 54929 -1.

54930 For *putmsg()*, if a control part is specified and *flags* is set to RS\_HIPRI, a high priority message  
 54931 shall be sent. If no control part is specified, and *flags* is set to RS\_HIPRI, *putmsg()* shall fail and  
 54932 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be  
 54933 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be  
 54934 sent and 0 shall be returned.

54935 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following  
 54936 mutually-exclusive flags defined: MSG\_HIPRI and MSG\_BAND. If *flags* is set to 0, *putpmsg()*  
 54937 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG\_HIPRI  
 54938 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG\_HIPRI and either  
 54939 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to  
 54940 [EINVAL]. If *flags* is set to MSG\_BAND, then a message shall be sent in the priority band  
 54941 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG\_BAND,  
 54942 no message shall be sent and 0 shall be returned.

**putmsg()**

54943 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control  
54944 conditions, with the following exceptions:

- 54945 • For high-priority messages, *putmsg()* shall not block on this condition and continues  
54946 processing the message.
- 54947 • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and  
54948 *O\_NONBLOCK* is set.

54949 The *putmsg()* function shall also block, unless prevented by lack of internal resources, while  
54950 waiting for the availability of message blocks in the STREAM, regardless of priority or whether  
54951 *O\_NONBLOCK* has been specified. No partial message shall be sent.

**RETURN VALUE**

54952 Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return  
54953 -1 and set *errno* to indicate the error.  
54954

**ERRORS**

54955 The *putmsg()* and *putpmsg()* functions shall fail if:  
54956

- |       |                  |  |
|-------|------------------|--|
| 54957 | [EAGAIN]         | A non-priority message was specified, the <i>O_NONBLOCK</i> flag is set, and the STREAM write queue is full due to internal flow control conditions; or buffers could not be allocated for the message that was to be created.   |
| 54958 |                  |  |
| 54959 |                  |  |
| 54960 | [EBADF]          | <i>fildev</i> is not a valid file descriptor open for writing.   |
| 54961 | [EINTR]          | A signal was caught during <i>putmsg()</i> .   |
| 54962 | [EINVAL]         | An undefined value is specified in <i>flags</i> , or <i>flags</i> is set to <i>RS_HIPRI</i> or <i>MSG_HIPRI</i> and no control part is supplied, or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer, or <i>flags</i> is set to <i>MSG_HIPRI</i> and <i>band</i> is non-zero (for <i>putpmsg()</i> only).                                |
| 54963 |                  |  |
| 54964 |                  |  |
| 54965 |                  |  |
| 54966 |                  |  |
| 54967 | [ENOSR]          | Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.  |
| 54968 |                  |  |
| 54969 | [ENOSTR]         | A STREAM is not associated with <i>fildev</i> .  |
| 54970 | [ENXIO]          | A hangup condition was generated downstream for the specified STREAM.  |
| 54971 | [EPIPE] or [EIO] | The <i>fildev</i> argument refers to a STREAMS-based pipe and the other end of the pipe is closed. A SIGPIPE signal is generated for the calling thread.   |
| 54972 |                  |  |
| 54973 | [ERANGE]         | The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message. |
| 54974 |                  |  |
| 54975 |                  |  |
| 54976 |                  |  |
| 54977 |                  |  |
| 54978 |                  |  |

54979 In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an  
54980 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
54981 *putmsg()* or *putpmsg()*, but reflects the prior error.

54982 **EXAMPLES**54983 **Sending a High-Priority Message**

54984 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the  
54985 following:

- 54986 1. Creates a high-priority message with a control part and a data part, using the buffers  
54987 pointed to by *ctrlbuf* and *databuf*, respectively.
- 54988 2. Sends the message to the STREAMS file identified by *fd*.

```
54989 #include <stropts.h>
54990 #include <string.h>
54991 ...
54992 int fd;
54993 char *ctrlbuf = "This is the control part";
54994 char *databuf = "This is the data part";
54995 struct strbuf ctrl;
54996 struct strbuf data;
54997 int ret;

54998 ctrl.buf = ctrlbuf;
54999 ctrl.len = strlen(ctrlbuf);

55000 data.buf = databuf;
55001 data.len = strlen(databuf);

55002 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

55003 **Using putpmsg()**

55004 This example has the same effect as the previous example. In this example, however, the  
55005 *putpmsg()* function creates and sends the message to the STREAMS file.

```
55006 #include <stropts.h>
55007 #include <string.h>
55008 ...
55009 int fd;
55010 char *ctrlbuf = "This is the control part";
55011 char *databuf = "This is the data part";
55012 struct strbuf ctrl;
55013 struct strbuf data;
55014 int ret;

55015 ctrl.buf = ctrlbuf;
55016 ctrl.len = strlen(ctrlbuf);

55017 data.buf = databuf;
55018 data.len = strlen(databuf);

55019 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

55020 **APPLICATION USAGE**

55021 None.

**putmsg()**55022 **RATIONALE**

55023 None.

55024 **FUTURE DIRECTIONS**55025 The *putmsg()* and *putpmsg()* functions may be removed in a future version.55026 **SEE ALSO**55027 [Section 2.6](#) (on page 494), *getmsg()*, *poll()*, *read()*, *write()*55028 XBD [<stropts.h>](#)55029 **CHANGE HISTORY**

55030 First released in Issue 4, Version 2.

55031 **Issue 5**

55032 Moved from X/OPEN UNIX extension to BASE.

55033 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the  
55034 control part of a message generated by *putmsg()* is at least 64 bytes in length”.55035 **Issue 6**

55036 This function is marked as part of the XSI STREAMS Option Group.

55037 The normative text is updated to avoid use of the term “must” for application requirements.

55038 **Issue 7**55039 The *putmsg()* and *putpmsg()* functions are marked obsolescent.

55040 **NAME**

55041 puts — put a string on standard output

55042 **SYNOPSIS**

55043 #include &lt;stdio.h&gt;

55044 int puts(const char \*s);

55045 **DESCRIPTION**

55046 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 55047 conflict between the requirements described here and the ISO C standard is unintentional. This  
 55048 volume of POSIX.1-2008 defers to the ISO C standard.

55049 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the  
 55050 standard output stream *stdout*. The terminating null byte shall not be written.

55051 CX The last data modification and last file status change timestamps of the file shall be marked for  
 55052 update between the successful execution of *puts()* and the next successful completion of a call to  
 55053 *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

55054 **RETURN VALUE**

55055 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall  
 55056 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

55057 **ERRORS**55058 Refer to *fputc()*.55059 **EXAMPLES**55060 **Printing to Standard Output**

55061 The following example gets the current time, converts it to a string using *localtime()* and  
 55062 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to  
 55063 an event for which it is waiting.

```
55064 #include <time.h>
55065 #include <stdio.h>
55066 ...
55067 time_t now;
55068 int minutes_to_event;
55069 ...
55070 time(&now);
55071 printf("The time is ");
55072 puts(asctime(localtime(&now)));
55073 printf("There are %d minutes to the event.\n",
55074        minutes_to_event);
55075 ...
```

55076 **APPLICATION USAGE**55077 The *puts()* function appends a <newline>, while *fputs()* does not.55078 **RATIONALE**

55079 None.

**puts()**55080 **FUTURE DIRECTIONS**

55081 None.

55082 **SEE ALSO**55083 *fopen()*, *fputs()*, *putc()*55084 XBD [<stdio.h>](#)55085 **CHANGE HISTORY**

55086 First released in Issue 1. Derived from Issue 1 of the SVID.

55087 **Issue 6**

55088 Extensions beyond the ISO C standard are marked.

55089 **Issue 7**

55090 Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55091 **NAME**

55092 pututxline — put an entry into the user accounting database

55093 **SYNOPSIS**

55094 XSI #include &lt;utmpx.h&gt;

55095 struct utmpx \*pututxline(const struct utmpx \*utmpx);

55096 **DESCRIPTION**55097 Refer to *endutxent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**putwc()**55098 **NAME**55099 `putwc` — put a wide character on a stream55100 **SYNOPSIS**55101 `#include <stdio.h>`55102 `#include <wchar.h>`55103 `wint_t putwc(wchar_t wc, FILE *stream);`55104 **DESCRIPTION**

55105 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 55106 conflict between the requirements described here and the ISO C standard is unintentional. This  
 55107 volume of POSIX.1-2008 defers to the ISO C standard.

55108 The `putwc()` function shall be equivalent to `fputwc()`, except that if it is implemented as a macro  
 55109 it may evaluate `stream` more than once, so the argument should never be an expression with  
 55110 side-effects.

55111 **RETURN VALUE**55112 Refer to `fputwc()`.55113 **ERRORS**55114 Refer to `fputwc()`.55115 **EXAMPLES**

55116 None.

55117 **APPLICATION USAGE**

55118 Since it may be implemented as a macro, `putwc()` may treat a `stream` argument with side-effects  
 55119 incorrectly. In particular, `putwc(wc,*f++)` need not work correctly. Therefore, use of this function  
 55120 is not recommended; `fputwc()` should be used instead.

55121 **RATIONALE**

55122 None.

55123 **FUTURE DIRECTIONS**

55124 None.

55125 **SEE ALSO**55126 `fputwc()`55127 XBD `<stdio.h>`, `<wchar.h>`55128 **CHANGE HISTORY**

55129 First released as a World-wide Portability Interface in Issue 4.

55130 **Issue 5**

55131 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument `wc`  
 55132 is changed from `wint_t` to `wchar_t`.

55133 The Optional Header (OH) marking is removed from `<stdio.h>`.

55134 **NAME**

55135 putwchar — put a wide character on a stdout stream

55136 **SYNOPSIS**

55137 #include &lt;wchar.h&gt;

55138 wint\_t putwchar(wchar\_t wc);

55139 **DESCRIPTION**

55140 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 55141 conflict between the requirements described here and the ISO C standard is unintentional. This  
 55142 volume of POSIX.1-2008 defers to the ISO C standard.

55143 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.55144 **RETURN VALUE**55145 Refer to *fputwc()*.55146 **ERRORS**55147 Refer to *fputwc()*.55148 **EXAMPLES**

55149 None.

55150 **APPLICATION USAGE**

55151 None.

55152 **RATIONALE**

55153 None.

55154 **FUTURE DIRECTIONS**

55155 None.

55156 **SEE ALSO**55157 *fputwc()*, *putwc()*

55158 XBD &lt;wchar.h&gt;

55159 **CHANGE HISTORY**

55160 First released in Issue 4

55161 **Issue 5**

55162 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
 55163 is changed from **wint\_t** to **wchar\_t**.

**pwrite()**55164 **NAME**

55165            pwrite — write on a file

55166 **SYNOPSIS**

55167            #include &lt;unistd.h&gt;

55168            ssize\_t pwrite(int *fildev*, const void \**buf*, size\_t *nbyte*,55169                    off\_t *offset*);55170 **DESCRIPTION**55171            Refer to *write()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55172 **NAME**

55173 qsort — sort a table of data

55174 **SYNOPSIS**

55175 #include &lt;stdlib.h&gt;

55176 void qsort(void \*base, size\_t nel, size\_t width,  
55177 int (\*compar)(const void \*, const void \*));55178 **DESCRIPTION**55179 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
55180 conflict between the requirements described here and the ISO C standard is unintentional. This  
55181 volume of POSIX.1-2008 defers to the ISO C standard.55182 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by  
55183 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has  
55184 the value zero, the comparison function pointed to by *compar* shall not be called and no  
55185 rearrangement shall take place.55186 The application shall ensure that the comparison function pointed to by *compar* does not alter the  
55187 contents of the array. The implementation may reorder elements of the array between calls to the  
55188 comparison function, but shall not alter the contents of any individual element.55189 When the same objects (consisting of *width* bytes, irrespective of their current positions in the  
55190 array) are passed more than once to the comparison function, the results shall be consistent with  
55191 one another. That is, they shall define a total ordering on the array.55192 The contents of the array shall be sorted in ascending order according to a comparison function.  
55193 The *compar* argument is a pointer to the comparison function, which is called with two  
55194 arguments that point to the elements being compared. The application shall ensure that the  
55195 function returns an integer less than, equal to, or greater than 0, if the first argument is  
55196 considered respectively less than, equal to, or greater than the second. If two members compare  
55197 as equal, their order in the sorted array is unspecified.55198 **RETURN VALUE**55199 The *qsort()* function shall not return a value.55200 **ERRORS**

55201 No errors are defined.

55202 **EXAMPLES**

55203 None.

55204 **APPLICATION USAGE**55205 The comparison function need not compare every byte, so arbitrary data may be contained in  
55206 the elements in addition to the values being compared.55207 **RATIONALE**55208 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a  
55209 pointer to elements of the array implies that for every call, for each argument separately, all of  
55210 the following expressions are non-zero:

55211 ((char \*)p - (char \*)base) % width == 0

55212 (char \*)p &gt;= (char \*)base

55213 (char \*)p &lt; (char \*)base + nel \* width

**qsort()**55214 **FUTURE DIRECTIONS**

55215 None.

55216 **SEE ALSO**55217 *alphasort()*

55218 XBD &lt;stdlib.h&gt;

55219 **CHANGE HISTORY**

55220 First released in Issue 1. Derived from Issue 1 of the SVID.

55221 **Issue 6**

55222 The normative text is updated to avoid use of the term “must” for application requirements.

55223 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to  
55224 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The  
55225 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55226 **NAME**

55227 raise — send a signal to the executing process

55228 **SYNOPSIS**

55229 #include &lt;signal.h&gt;

55230 int raise(int sig);

55231 **DESCRIPTION**55232 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
55233 conflict between the requirements described here and the ISO C standard is unintentional. This  
55234 volume of POSIX.1-2008 defers to the ISO C standard.55235 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal  
55236 handler is called, the *raise()* function shall not return until after the signal handler does.55237 CX The effect of the *raise()* function shall be equivalent to calling:

55238 pthread\_kill(pthread\_self(), sig);

55239 **RETURN VALUE**55240 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned  
55241 and *errno* shall be set to indicate the error.55242 **ERRORS**55243 The *raise()* function shall fail if:55244 CX [EINVAL] The value of the *sig* argument is an invalid signal number.55245 **EXAMPLES**

55246 None.

55247 **APPLICATION USAGE**

55248 None.

55249 **RATIONALE**

55250 The term “thread” is an extension to the ISO C standard.

55251 **FUTURE DIRECTIONS**

55252 None.

55253 **SEE ALSO**55254 *kill()*, *sigaction()*

55255 XBD &lt;signal.h&gt;, &lt;sys/types.h&gt;

55256 **CHANGE HISTORY**

55257 First released in Issue 4. Derived from the ANSI C standard.

55258 **Issue 5**

55259 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

55260 **Issue 6**

55261 Extensions beyond the ISO C standard are marked.

55262 The following new requirements on POSIX implementations derive from alignment with the  
55263 Single UNIX Specification:

- 55264
- In the RETURN VALUE section, the requirement to set *errno* on error is added.

## raise()

55265

- The [EINVAL] error condition is added.

55266 **Issue 7**

55267

Functionality relating to the Threads option is moved to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55268 **NAME**

55269 rand, rand\_r, srand — pseudo-random number generator

55270 **SYNOPSIS**

```
55271     #include <stdlib.h>
55272     int rand(void);
55273 OB CX  int rand_r(unsigned *seed);
55274     void srand(unsigned seed);
```

55275 **DESCRIPTION**

55276 CX For *rand()* and *srand()*: The functionality described on this reference page is aligned with the  
 55277 ISO C standard. Any conflict between the requirements described here and the ISO C standard is  
 55278 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

55279 The *rand()* function shall compute a sequence of pseudo-random integers in the range  
 55280 XSI [0,{RAND\_MAX}] with a period of at least  $2^{32}$ .

55281 CX The *rand()* function need not be thread-safe.

55282 OB CX The *rand\_r()* function shall compute a sequence of pseudo-random integers in the range  
 55283 [0,{RAND\_MAX}]. (The value of the {RAND\_MAX} macro shall be at least 32767.)

55284 If *rand\_r()* is called with the same initial value for the object pointed to by *seed* and that object is  
 55285 not modified between successive returns and calls to *rand\_r()*, the same sequence shall be  
 55286 generated.

55287 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random  
 55288 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same  
 55289 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before  
 55290 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first  
 55291 called with a seed value of 1.

55292 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
 55293 *rand()* or *srand()*.

55294 **RETURN VALUE**

55295 The *rand()* function shall return the next pseudo-random number in the sequence.

55296 OB CX The *rand\_r()* function shall return a pseudo-random integer.

55297 The *srand()* function shall not return a value.

55298 **ERRORS**

55299 No errors are defined.

55300 **EXAMPLES**55301 **Generating a Pseudo-Random Number Sequence**

55302 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
55303     #include <stdio.h>
55304     #include <stdlib.h>
55305     ...
55306     long count, i;
55307     char *keyst;
55308     int elementlen, len;
55309     char c;
55310     ...
```

**rand()**

```

55311     /* Initial random number generator. */
55312     srand(1);

55313     /* Create keys using only lowercase characters */
55314     len = 0;
55315     for (i=0; i<count; i++) {
55316         while (len < elementlen) {
55317             c = (char) (rand() % 128);
55318             if (islower(c))
55319                 keystr[len++] = c;
55320         }

55321         keystr[len] = '\0';
55322         printf("%s Element%0*ld\n", keystr, elementlen, i);
55323         len = 0;
55324     }

```

55325 **Generating the Same Sequence on Different Machines**

55326 The following code defines a pair of functions that could be incorporated into applications  
55327 wishing to ensure that the same sequence of numbers is generated across different machines.

```

55328     static unsigned long next = 1;
55329     int myrand(void) /* RAND_MAX assumed to be 32767. */
55330     {
55331         next = next * 1103515245 + 12345;
55332         return((unsigned)(next/65536) % 32768);
55333     }

55334     void mysrand(unsigned seed)
55335     {
55336         next = seed;
55337     }

```

55338 **APPLICATION USAGE**

55339 The *drand48()* function provides a much more elaborate random number generator.

55340 The limitations on the amount of state that can be carried between one function call and another  
55341 mean the *rand\_r()* function can never be implemented in a way which satisfies all of the  
55342 requirements on a pseudo-random number generator. Therefore this function should be avoided  
55343 whenever non-trivial requirements (including safety) have to be fulfilled.

55344 **RATIONALE**

55345 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams  
55346 shared by all threads. Those two functions need not change, but there has to be mutual-  
55347 exclusion that prevents interference between two threads concurrently accessing the random  
55348 number generator.

55349 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded  
55350 program:

- 55351 1. A single per-process sequence of pseudo-random numbers that is shared by all threads  
55352 that call *rand()*
- 55353 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

55354 This is provided by the modified thread-safe function based on whether the seed value is global

55355 to the entire process or local to each thread.

55356 This does not address the known deficiencies of the *rand()* function implementations, which  
55357 have been approached by maintaining more state. In effect, this specifies new thread-safe forms  
55358 of a deficient function.

55359 **FUTURE DIRECTIONS**

55360 The *rand\_r()* function may be removed in a future version.

55361 **SEE ALSO**

55362 *drand48()*

55363 XBD <stdlib.h>

55364 **CHANGE HISTORY**

55365 First released in Issue 1. Derived from Issue 1 of the SVID.

55366 **Issue 5**

55367 The *rand\_r()* function is included for alignment with the POSIX Threads Extension.

55368 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.

55369 **Issue 6**

55370 Extensions beyond the ISO C standard are marked.

55371 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

55372 **Issue 7**

55373 Austin Group Interpretation 1003.1-2001 #156 is applied.

55374 The *rand\_r()* function is marked obsolescent.

**random()***System Interfaces*55375 **NAME**

55376 random — generate pseudo-random number

55377 **SYNOPSIS**55378 XSI 

```
#include <stdlib.h>
```

  
55379 

```
long random(void);
```

55380 **DESCRIPTION**55381 Refer to *initstate()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55382 **NAME**

55383        pread, read — read from a file

55384 **SYNOPSIS**

55385        #include &lt;unistd.h&gt;

55386        ssize\_t pread(int *fildes*, void \**buf*, size\_t *nbyte*, off\_t *offset*);55387        ssize\_t read(int *fildes*, void \**buf*, size\_t *nbyte*);55388 **DESCRIPTION**

55389        The *read()* function shall attempt to read *nbyte* bytes from the file associated with the open file  
 55390        descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on  
 55391        the same pipe, FIFO, or terminal device is unspecified.

55392        Before any action described below is taken, and if *nbyte* is zero, the *read()* function may detect  
 55393        and return errors as described below. In the absence of errors, or if error detection is not  
 55394        performed, the *read()* function shall return zero and have no other results.

55395        On files that support seeking (for example, a regular file), the *read()* shall start at a position in  
 55396        the file given by the file offset associated with *fildes*. The file offset shall be incremented by the  
 55397        number of bytes actually read.

55398        Files that do not support seeking—for example, terminals—always read from the current  
 55399        position. The value of a file offset associated with such a file is undefined.

55400        No data transfer shall occur past the current end-of-file. If the starting position is at or after the  
 55401        end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent  
 55402        *read()* requests is implementation-defined.

55403        If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

55404        When attempting to read from an empty pipe or FIFO:

- 55405        • If no process has the pipe open for writing, *read()* shall return 0 to indicate end-of-file.
- 55406        • If some process has the pipe open for writing and O\_NONBLOCK is set, *read()* shall return  
 55407        -1 and set *errno* to [EAGAIN].
- 55408        • If some process has the pipe open for writing and O\_NONBLOCK is clear, *read()* shall  
 55409        block the calling thread until some data is written or the pipe is closed by all processes that  
 55410        had the pipe open for writing.

55411        When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and  
 55412        has no data currently available:

- 55413        • If O\_NONBLOCK is set, *read()* shall return -1 and set *errno* to [EAGAIN].
- 55414        • If O\_NONBLOCK is clear, *read()* shall block the calling thread until some data becomes  
 55415        available.
- 55416        • The use of the O\_NONBLOCK flag has no effect if there is some data available.

55417        The *read()* function reads data previously written to a file. If any portion of a regular file prior to  
 55418        the end-of-file has not been written, *read()* shall return bytes with value 0. For example, *lseek()*  
 55419        allows the file offset to be set beyond the end of existing data in the file. If data is later written at  
 55420        this point, subsequent reads in the gap between the previous end of data and the newly written  
 55421        data shall return bytes with value 0 until data is written into the gap.

55422        Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the last  
 55423        data access timestamp of the file, and shall return the number of bytes read. This number shall  
 55424        never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

**read()**

- 55425 left in the file is less than *nbyte*, if the *read()* request was interrupted by a signal, or if the file is a  
 55426 pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading.  
 55427 For example, a *read()* from a file associated with a terminal may return one typed line of data.
- 55428 If a *read()* is interrupted by a signal before it reads any data, it shall return  $-1$  with *errno* set to  
 55429 `[EINTR]`.
- 55430 If a *read()* is interrupted by a signal after it has successfully read some data, it shall return the  
 55431 number of bytes read.
- 55432 For regular files, no data transfer shall occur past the offset maximum established in the open  
 55433 file description associated with *fildes*.
- 55434 If *fildes* refers to a socket, *read()* shall be equivalent to *recv()* with no flags set.
- 55435 SIO If the `O_DSYNC` and `O_RSYNC` bits have been set, read I/O operations on the file descriptor  
 55436 shall complete as defined by synchronized I/O data integrity completion. If the `O_SYNC` and  
 55437 `O_RSYNC` bits have been set, read I/O operations on the file descriptor shall complete as  
 55438 defined by synchronized I/O file integrity completion.
- 55439 SHM If *fildes* refers to a shared memory object, the result of the *read()* function is unspecified.
- 55440 TYM If *fildes* refers to a typed memory object, the result of the *read()* function is unspecified.
- 55441 OB XSR A *read()* from a STREAMS file can read data in three different modes: *byte-stream* mode, *message-*  
 55442 *nondiscard* mode, and *message-discard* mode. The default shall be *byte-stream* mode. This can be  
 55443 changed using the `I_SRDOPT ioctl()` request, and can be tested with `I_GRDOPT ioctl()`. In *byte-*  
 55444 *stream* mode, *read()* shall retrieve data from the STREAM until as many bytes as were requested  
 55445 are transferred, or until there is no more data to be retrieved. *Byte-stream* mode ignores  
 55446 message boundaries.
- 55447 In STREAMS *message-nondiscard* mode, *read()* shall retrieve data until as many bytes as were  
 55448 requested are transferred, or until a message boundary is reached. If *read()* does not retrieve all  
 55449 the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by  
 55450 the next *read()* call. *Message-discard* mode also retrieves data until as many bytes as were  
 55451 requested are transferred, or a message boundary is reached. However, unread data remaining  
 55452 in a message after the *read()* returns shall be discarded, and shall not be available for a  
 55453 subsequent *read()*, *getmsg()*, or *getpmsg()* call.
- 55454 How *read()* handles zero-byte STREAMS messages is determined by the current read mode  
 55455 setting. In *byte-stream* mode, *read()* shall accept data until it has read *nbyte* bytes, or until there  
 55456 is no more data to read, or until a zero-byte message block is encountered. The *read()* function  
 55457 shall then return the number of bytes read, and place the zero-byte message back on the  
 55458 STREAM to be retrieved by the next *read()*, *getmsg()*, or *getpmsg()*. In *message-nondiscard*  
 55459 mode or *message-discard* mode, a zero-byte message shall return 0 and the message shall be  
 55460 removed from the STREAM. When a zero-byte message is read as the first message on a  
 55461 STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless  
 55462 of the read mode.
- 55463 A *read()* from a STREAMS file shall return the data in the message at the front of the STREAM  
 55464 head read queue, regardless of the priority band of the message.
- 55465 By default, STREAMS are in *control-normal* mode, in which a *read()* from a STREAMS file can  
 55466 only process messages that contain a data part but do not contain a control part. The *read()* shall  
 55467 fail if a message containing a control part is encountered at the STREAM head. This default  
 55468 action can be changed by placing the STREAM in either *control-data* mode or *control-discard*  
 55469 mode with the `I_SRDOPT ioctl()` command. In *control-data* mode, *read()* shall convert any  
 55470 control part to data and pass it to the application before passing any data part originally present

55471 in the same message. In control-discard mode, *read()* shall discard message control parts but  
55472 return to the process any data part in the message.

55473 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before  
55474 the call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflect the prior  
55475 error. If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally  
55476 until the STREAM head read queue is empty. Thereafter, it shall return 0.

55477 The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position  
55478 in the file without changing the file pointer. The first three arguments to *pread()* are the same as  
55479 *read()* with the addition of a fourth argument *offset* for the desired position inside the file. An  
55480 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

#### 55481 RETURN VALUE

55482 Upon successful completion, these functions shall return a non-negative integer indicating the  
55483 number of bytes actually read. Otherwise, the functions shall return -1 and set *errno* to indicate  
55484 the error.

#### 55485 ERRORS

55486 These functions shall fail if:

55487 [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the thread would be  
55488 delayed.

55489 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

55490 OB XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message  
55491 waiting to be read includes a control part.

55492 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
55493 was transferred.

55494 OB XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
55495 indirectly) downstream from a multiplexer.

55496 [EIO] The process is a member of a background process group attempting to read  
55497 from its controlling terminal, the process is ignoring or blocking the SIGTTIN  
55498 signal, or the process group is orphaned. This error may also be generated for  
55499 implementation-defined reasons.

55500 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not  
55501 allow the directory to be read using *read()* or *pread()*. The *readdir()* function  
55502 should be used instead.

55503 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before  
55504 the end-of-file, and the starting position is greater than or equal to the offset  
55505 maximum established in the open file description associated with *fildev*.

55506 The *read()* function shall fail if:

55507 [EAGAIN] or [EWOULDBLOCK]

55508 The file descriptor is for a socket, is marked O\_NONBLOCK, and no data is  
55509 waiting to be received.

55510 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by  
55511 its peer.

55512 [ENOTCONN] A read was attempted on a socket that is not connected.

**read()**

55513	[ETIMEDOUT]	A read was attempted on a socket and a transmission timeout occurred.
55514		These functions may fail if:
55515	[EIO]	A physical I/O error has occurred.
55516	[ENOBUFFS]	Insufficient resources were available in the system to perform the operation.
55517	[ENOMEM]	Insufficient memory was available to fulfill the request.
55518	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
55519		
55520		The <i>pread()</i> function shall fail, and the file pointer shall remain unchanged, if:
55521	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.
55522	[EOVERFLOW]	The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the file.
55523		
55524	[ENXIO]	A request was outside the capabilities of the device.
55525	[ESPIPE]	<i>fildev</i> is associated with a pipe or FIFO.

55526 **EXAMPLES**55527 **Reading Data into a Buffer**

55528 The following example reads data from the file associated with the file descriptor *fd* into the  
55529 buffer pointed to by *buf*.

```
55530 #include <sys/types.h>
55531 #include <unistd.h>
55532 ...
55533 char buf[20];
55534 size_t nbytes;
55535 ssize_t bytes_read;
55536 int fd;
55537 ...
55538 nbytes = sizeof(buf);
55539 bytes_read = read(fd, buf, nbytes);
55540 ...
```

55541 **APPLICATION USAGE**

55542 None.

55543 **RATIONALE**

55544 This volume of POSIX.1-2008 does not specify the value of the file offset after an error is  
55545 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
55546 meaningless since no file is involved. For errors that are detected immediately, such as  
55547 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
55548 an updated value would be very useful and is the behavior of many implementations.

55549 Note that a *read()* of zero bytes does not modify the last data access timestamp. A *read()* that  
55550 requests more than zero bytes, but returns zero, is required to modify the last data access  
55551 timestamp.

55552 Implementations are allowed, but not required, to perform error checking for *read()* requests of  
55553 zero bytes.

55554 **Input and Output**

55555 The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and  
 55556 *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use  
 55557 abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be  
 55558 declared so that existing functions work, but can also be declared so that larger types can be  
 55559 represented in future implementations. It is presumed that whatever constraints limit the  
 55560 maximum range of **size\_t** also limit portable I/O requests to the same range. This volume of  
 55561 POSIX.1-2008 also limits the range further by requiring that the byte count be limited so that a  
 55562 signed return value remains meaningful. Since the return type is also a (signed) abstract type,  
 55563 the byte count can be defined by the implementation to be larger than an **int** can hold.

55564 The standard developers considered adding atomicity requirements to a pipe or FIFO, but  
 55565 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of  
 55566 reads of {PIPE\_BUF} or any other size that would be an aid to applications portability.

55567 This volume of POSIX.1-2008 requires that no action be taken for *read()* or *write()* when *nbyte* is  
 55568 zero. This is not intended to take precedence over detection of errors (such as invalid buffer  
 55569 pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2008, but  
 55570 the phrasing here could be misread to require detection of the zero case before any other errors.  
 55571 A value of zero is to be considered a correct value, for which the semantics are a no-op.

55572 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the  
 55573 bytes from a single operation that started out together end up together, without interleaving  
 55574 from other I/O operations. It is a known attribute of terminals that this is not honored, and  
 55575 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.  
 55576 The behavior for other device types is also left unspecified, but the wording is intended to imply  
 55577 that future standards might choose to specify atomicity (or not).

55578 There were recommendations to add format parameters to *read()* and *write()* in order to handle  
 55579 networked transfers among heterogeneous file system and base hardware types. Such a facility  
 55580 may be required for support by the OSI presentation of layer services. However, it was  
 55581 determined that this should correspond with similar C-language facilities, and that is beyond  
 55582 the scope of this volume of POSIX.1-2008. The concept was suggested to the developers of the  
 55583 ISO C standard for their consideration as a possible area for future work.

55584 In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does  
 55585 not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth  
 55586 Edition, there is an additional function, *select()*, whose purpose is to pause until specified  
 55587 activity (data to read, space to write, and so on) is detected on specified file descriptors. It is  
 55588 common in applications written for those systems for *select()* to be used before *read()* in  
 55589 situations (such as keyboard input) where interruption of I/O due to a signal is desired.

55590 The issue of which files or file types are interruptible is considered an implementation design  
 55591 issue. This is often affected primarily by hardware and reliability issues.

55592 There are no references to actions taken following an “unrecoverable error”. It is considered  
 55593 beyond the scope of this volume of POSIX.1-2008 to describe what happens in the case of  
 55594 hardware errors.

55595 Earlier versions of this standard allowed two very different behaviors with regard to the  
 55596 handling of interrupts. In order to minimize the resulting confusion, it was decided that  
 55597 POSIX.1-2008 should support only one of these behaviors. Historical practice on AT&T-derived  
 55598 systems was to have *read()* and *write()* return  $-1$  and set *errno* to [EINTR] when interrupted after  
 55599 some, but not all, of the data requested had been transferred. However, the US Department of  
 55600 Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read()* and

**read()**

55601 *write()* return the number of bytes actually transferred before the interrupt. If  $-1$  is returned  
 55602 when any data is transferred, it is difficult to recover from the error on a seekable device and  
 55603 impossible on a non-seekable device. Most new implementations support this behavior. The  
 55604 behavior required by POSIX.1-2008 is to return the number of bytes transferred.

55605 POSIX.1-2008 does not specify when an implementation that buffers *read()*s actually moves the  
 55606 data into the user-supplied buffer, so an implementation may choose to do this at the latest  
 55607 possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a  
 55608 partial byte count, but rather to return  $-1$  and set *errno* to [EINTR].

55609 Consideration was also given to combining the two previous options, and setting *errno* to  
 55610 [EINTR] while returning a short count. However, not only is there no existing practice that  
 55611 implements this, it is also contradictory to the idea that when *errno* is set, the function  
 55612 responsible shall return  $-1$ .

55613 **FUTURE DIRECTIONS**

55614 None.

55615 **SEE ALSO**

55616 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*

55617 XBD Chapter 11 (on page 199), *<stropts.h>*, *<sys/uio.h>*, *<unistd.h>*

55618 **CHANGE HISTORY**

55619 First released in Issue 1. Derived from Issue 1 of the SVID.

55620 **Issue 5**

55621 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 55622 Threads Extension.

55623 Large File Summit extensions are added.

55624 The *pread()* function is added.

55625 **Issue 6**

55626 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 55627 marked as part of the XSI STREAMS Option Group.

55628 The following new requirements on POSIX implementations derive from alignment with the  
 55629 Single UNIX Specification:

55630 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has  
 55631 successfully read some data, it returns the number of bytes read. In Issue 3, it was optional  
 55632 whether *read()* returned the number of bytes read, or whether it returned  $-1$  with *errno* set  
 55633 to [EINTR]. This is a FIPS requirement.

55634 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer  
 55635 occurs past the offset maximum established in the open file description associated with  
 55636 *files*. This change is to support large files.

55637 • The [Eoverflow] mandatory error condition is added.

55638 • The [ENxio] optional error condition is added.

55639 Text referring to sockets is added to the DESCRIPTION.

55640 The following changes were made to align with the IEEE P1003.1a draft standard:

55641 • The effect of reading zero bytes is clarified.

55642 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that

- 55643 *read()* results are unspecified for typed memory objects.
- 55644 New RATIONALE is added to explain the atomicity requirements for input and output  
55645 operations.
- 55646 The following error conditions are added for operations on sockets: [EAGAIN],  
55647 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].
- 55648 The [EIO] error is made optional.
- 55649 The following error conditions are added for operations on sockets: [ENOBUFS] and  
55650 [ENOMEM].
- 55651 The *readv()* function is split out into a separate reference page.
- 55652 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN]  
55653 error in the ERRORS section from “the process would be delayed” to “the thread would be  
55654 delayed”.
- 55655 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial  
55656 correction in the RATIONALE section.
- 55657 **Issue 7**
- 55658 The *pread()* function is moved from the XSI option to the Base.
- 55659 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 55660 Changes are made related to support for finegrained timestamps.

**readdir()**55661 **NAME**

55662 readdir, readdir\_r — read a directory

55663 **SYNOPSIS**

```
55664 #include <dirent.h>
55665 struct dirent *readdir(DIR *dirp);
55666 int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
55667 struct dirent **restrict result);
```

55668 **DESCRIPTION**

55669 The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is  
 55670 an ordered sequence of all the directory entries in a particular directory. Directory entries  
 55671 represent files; files may be removed from a directory or added to a directory asynchronously to  
 55672 the operation of *readdir()*.

55673 The *readdir()* function shall return a pointer to a structure representing the directory entry at the  
 55674 current position in the directory stream specified by the argument *dirp*, and position the  
 55675 directory stream at the next entry. It shall return a null pointer upon reaching the end of the  
 55676 directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory  
 55677 entry. The value of the structure's *d\_ino* member shall be set to the file serial number of the file  
 55678 named by the *d\_name* member. If the *d\_name* member names a symbolic link, the value of the  
 55679 *d\_ino* member shall be set to the file serial number of the symbolic link itself.

55680 The *readdir()* function shall not return directory entries containing empty names. If entries for  
 55681 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-  
 55682 dot; otherwise, they shall not be returned.

55683 The pointer returned by *readdir()* points to data which may be overwritten by another call to  
 55684 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*  
 55685 on a different directory stream.

55686 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 55687 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

55688 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*  
 55689 shall mark for update the last data access timestamp of the directory each time the directory is  
 55690 actually read.

55691 After a call to *fork()*, either the parent or child (but not both) may continue processing the  
 55692 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes  
 55693 use these functions, the result is undefined.

55694 The *readdir()* function need not be thread-safe.

55695 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If  
 55696 *errno* is set to non-zero on return, an error occurred.

55697 The *readdir\_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the  
 55698 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer  
 55699 to this structure at the location referenced by *result*, and position the directory stream at the next  
 55700 entry.

55701 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d\_name*  
 55702 members containing at least **[NAME\_MAX]+1** elements.

55703 Upon successful return, the pointer returned at *\*result* shall have the same value as the argument  
 55704 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value **NULL**.

55705 The *readdir\_r()* function shall not return directory entries containing empty names.

55706 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
55707 *rewinddir()*, whether a subsequent call to *readdir\_r()* returns an entry for that file is unspecified.

55708 The *readdir\_r()* function may buffer several directory entries per actual read operation;  
55709 *readdir\_r()* shall mark for update the last data access timestamp of the directory each time the  
55710 directory is actually read.

#### 55711 RETURN VALUE

55712 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.  
55713 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate  
55714 the error. When the end of the directory is encountered, a null pointer shall be returned and  
55715 *errno* is not changed.

55716 If successful, the *readdir\_r()* function shall return zero; otherwise, an error number shall be  
55717 returned to indicate the error.

#### 55718 ERRORS

55719 These functions shall fail if:

55720 [EOVERFLOW] One of the values in the structure to be returned cannot be represented  
55721 correctly.

55722 These functions may fail if:

55723 [EBADF] The *dirp* argument does not refer to an open directory stream.

55724 [ENOENT] The current position of the directory stream is invalid.

#### 55725 EXAMPLES

55726 The following sample program searches the current directory for each of the arguments supplied  
55727 on the command line.

```
55728 #include <dirent.h>
55729 #include <errno.h>
55730 #include <stdio.h>
55731 #include <string.h>

55732 static void lookup(const char *arg)
55733 {
55734     DIR *dirp;
55735     struct dirent *dp;

55736     if ((dirp = opendir(".")) == NULL) {
55737         perror("couldn't open '.'");
55738         return;
55739     }

55740     do {
55741         errno = 0;
55742         if ((dp = readdir(dirp)) != NULL) {
55743             if (strcmp(dp->d_name, arg) != 0)
55744                 continue;

55745             (void) printf("found %s\n", arg);
55746             (void) closedir(dirp);
55747             return;
55748         }
55749     } while (dp != NULL);
```

**readdir()**

```

55750         if (errno != 0)
55751             perror("error reading directory");
55752         else
55753             (void) printf("failed to find %s\n", arg);
55754             (void) closedir(dirp);
55755             return;
55756     }

55757     int main(int argc, char *argv[])
55758     {
55759         int i;
55760         for (i = 1; i < argc; i++)
55761             lookup(argv[i]);
55762         return (0);
55763     }

```

**55764 APPLICATION USAGE**

55765 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to  
 55766 examine the contents of the directory.

55767 The *readdir\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 55768 of possibly using a static data area that may be overwritten by each call.

**55769 RATIONALE**

55770 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be  
 55771 inferred.

55772 Historical implementations of *readdir()* obtain multiple directory entries on a single read  
 55773 operation, which permits subsequent *readdir()* operations to operate from the buffered  
 55774 information. Any wording that required each successful *readdir()* operation to mark the  
 55775 directory last data access timestamp for update would disallow such historical performance-  
 55776 oriented implementations.

55777 When returning a directory entry for the root of a mounted file system, some historical  
 55778 implementations of *readdir()* returned the file serial number of the underlying mount point,  
 55779 rather than of the root of the mounted file system. This behavior is considered to be a bug, since  
 55780 the underlying file serial number has no significance to applications.

55781 Since *readdir()* returns NULL when it detects an error and when the end of the directory is  
 55782 encountered, an application that needs to tell the difference must set *errno* to zero before the call  
 55783 and check if NULL is returned. Since the function must not change *errno* in the second case  
 55784 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL  
 55785 indicates end-of-directory; otherwise, an error.

55786 Routines to deal with this problem more directly were proposed:

```

55787 int derror (dirp)
55788 DIR *dirp;

55789 void clearderr (dirp)
55790 DIR *dirp;

```

55791 The first would indicate whether an error had occurred, and the second would clear the error  
 55792 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*  
 55793 not change *errno* when end-of-directory is encountered.

55794 An error or signal indicating that a directory has changed while open was considered but  
 55795 rejected.

55796 The thread-safe version of the directory reading function returns values in a user-supplied buffer  
 55797 instead of possibly using a static data area that may be overwritten by each call. Either the  
 55798 {NAME\_MAX} compile-time constant or the corresponding *pathconf()* option can be used to  
 55799 determine the maximum sizes of returned pathnames.

#### 55800 FUTURE DIRECTIONS

55801 None.

#### 55802 SEE ALSO

55803 *closedir()*, *dirfd()*, *exec*, *fdopendir()*, *fstatat()*, *rewinddir()*, *symlink()*

55804 XBD `<dirent.h>`, `<sys/types.h>`

#### 55805 CHANGE HISTORY

55806 First released in Issue 2.

#### 55807 Issue 5

55808 Large File Summit extensions are added.

55809 The *readdir\_r()* function is included for alignment with the POSIX Threads Extension.

55810 A note indicating that the *readdir()* function need not be reentrant is added to the  
 55811 DESCRIPTION.

#### 55812 Issue 6

55813 The *readdir\_r()* function is marked as part of the Thread-Safe Functions option.

55814 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.

55815 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful  
 55816 return for the *readdir\_r()* function.

55817 The following new requirements on POSIX implementations derive from alignment with the  
 55818 Single UNIX Specification:

- 55819 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 55820 required for conforming implementations of previous POSIX specifications, it was not  
 55821 required for UNIX applications.
- 55822 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
 55823 **struct dirent** when an entry refers to a symbolic link.
- 55824 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 55825 files.
- 55826 • The [ENOENT] optional error condition is added.

55827 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 55828 its avoidance of possibly using a static data area.

55829 The **restrict** keyword is added to the *readdir\_r()* prototype for alignment with the  
 55830 ISO/IEC 9899:1999 standard.

55831 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES  
 55832 section with a new example.

#### 55833 Issue 7

55834 Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

55835 Austin Group Interpretation 1003.1-2001 #156 is applied.

55836 The *readdir\_r()* function is moved from the Thread-Safe Functions option to the Base.

**readdir()**

- 55837 Changes are made related to support for finegrained timestamps.
- 55838 The value of the *d\_ino* member is no longer unspecified for symbolic links.
- 55839 SD5-XSH-ERN-193 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

55840 **NAME**

55841 readlink, readlinkat — read the contents of a symbolic link relative to a directory file descriptor

55842 **SYNOPSIS**

```
55843 #include <unistd.h>
55844 ssize_t readlink(const char *restrict path, char *restrict buf,
55845                 size_t bufsize);
55846 ssize_t readlinkat(int fd, const char *restrict path,
55847                  char *restrict buf, size_t bufsize);
```

55848 **DESCRIPTION**

55849 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the  
 55850 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,  
 55851 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to  
 55852 contain the link content, the first *bufsize* bytes shall be placed in *buf*.

55853 If the value of *bufsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

55854 Upon successful completion, *readlink()* shall mark for update the last data access timestamp of  
 55855 the symbolic link.

55856 The *readlinkat()* function shall be equivalent to the *readlink()* function except in the case where  
 55857 *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the  
 55858 directory associated with the file descriptor *fd* instead of the current working directory. If the file  
 55859 descriptor was opened without O\_SEARCH, the function shall check whether directory searches  
 55860 are permitted using the current permissions of the directory underlying the file descriptor. If the  
 55861 file descriptor was opened with O\_SEARCH, the function shall not perform the check.

55862 If *readlinkat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 55863 directory is used and the behavior shall be identical to a call to *readlink()*.

55864 **RETURN VALUE**

55865 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.  
 55866 Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the  
 55867 error.

55868 Upon successful completion, the *readlinkat()* function shall return 0. Otherwise, it shall return -1  
 55869 and set *errno* to indicate the error.

55870 **ERRORS**

55871 These functions shall fail if:

55872 [EACCES] Search permission is denied for a component of the path prefix of *path*.

55873 [EINVAL] The *path* argument names a file that is not a symbolic link.

55874 [EIO] An I/O error occurred while reading from the file system.

55875 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 55876 argument.

55877 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME\_MAX}.

55879 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

55880 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
 55881 contains at least one non-*<slash>* character and ends with one or more trailing  
 55882 *<slash>* characters and the last pathname component names an existing file  
 55883 that is neither a directory nor a symbolic link to a directory.

**readlink()**

- 55884 The *readlinkat()* function shall fail if:
- 55885 [EACCES] *fd* was not opened with O\_SEARCH and the permissions of the directory  
55886 underlying *fd* do not permit directory searches.
- 55887 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
55888 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.
- 55889 These functions may fail if:
- 55890 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
55891 resolution of the *path* argument.
- 55892 [ENAMETOOLONG]  
55893 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
55894 symbolic link produced an intermediate result with a length that exceeds  
55895 {PATH\_MAX}.
- 55896 The *readlinkat()* function may fail if:
- 55897 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
55898 file descriptor associated with a directory.

55899 **EXAMPLES**55900 **Reading the Name of a Symbolic Link**

55901 The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
55902 #include <unistd.h>
55903 char buf[1024];
55904 ssize_t len;
55905 ...
55906 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
55907     buf[len] = '\0';
```

55908 **APPLICATION USAGE**

55909 Conforming applications should not assume that the returned contents of the symbolic link are  
55910 null-terminated.

55911 **RATIONALE**

55912 Since POSIX.1-2008 does not require any association of file times with symbolic links, there is no  
55913 requirement that file times be updated by *readlink()*. The type associated with *bufsiz* is a **size\_t**  
55914 in order to be consistent with both the ISO C standard and the definition of *read()*. The behavior  
55915 specified for *readlink()* when *bufsiz* is zero represents historical practice. For this case, the  
55916 standard developers considered a change whereby *readlink()* would return the number of non-  
55917 null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however,  
55918 since the **stat** structure member *st\_size* value can be used to determine the size of buffer  
55919 necessary to contain the contents of the symbolic link as returned by *readlink()*, this proposal  
55920 was rejected, and the historical practice retained.

55921 The purpose of the *readlinkat()* function is to read the content of symbolic links in directories  
55922 other than the current working directory without exposure to race conditions. Any part of the  
55923 path of a file could be changed in parallel to a call to *readlink()*, resulting in unspecified behavior.  
55924 By opening a file descriptor for the target directory and using the *readlinkat()* function it can be  
55925 guaranteed that the symbolic link read is located relative to the desired directory.

55926 **FUTURE DIRECTIONS**

55927 None.

55928 **SEE ALSO**55929 *fstatat()*, *symlink()*55930 XBD <*unistd.h*>55931 **CHANGE HISTORY**

55932 First released in Issue 4, Version 2.

55933 **Issue 5**

55934 Moved from X/OPEN UNIX extension to BASE.

55935 **Issue 6**55936 The return type is changed to **ssize\_t**, to align with the IEEE P1003.1a draft standard.55937 The following new requirements on POSIX implementations derive from alignment with the  
55938 Single UNIX Specification:

- 55939 • This function is made mandatory.
- 55940 • In this function it is possible for the return value to exceed the range of the type **ssize\_t**  
55941 (since **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the  
55942 size of the **size\_t** object is added to the description to resolve this conflict.

55943 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 55944 • The FUTURE DIRECTIONS section is changed to None.

55945 The following changes were made to align with the IEEE P1003.1a draft standard:

- 55946 • The [ELOOP] optional error condition is added.

55947 The **restrict** keyword is added to the *readlink()* prototype for alignment with the  
55948 ISO/IEC 9899: 1999 standard.55949 **Issue 7**

55950 Austin Group Interpretation 1003.1-2001 #143 is applied.

55951 SD5-XSH-ERN-189 is applied, updating the ERRORS section.

55952 The *readlinkat()* function is added from The Open Group Technical Standard, 2006, Extended  
55953 API Set Part 2.

55954 The [EACCES] error is removed from the “may fail” error conditions.

55955 Changes are made to allow a directory to be opened for searching.

55956 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
55957 pathname exists but is not a directory or a symbolic link to a directory.

**readv()**55958 **NAME**

55959 readv — read a vector

55960 **SYNOPSIS**

```
55961 xSI #include <sys/uio.h>
55962      ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

55963 **DESCRIPTION**

55964 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*  
 55965 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*  
 55966 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than  
 55967 or equal to {IOV\_MAX}.

55968 Each *iovec* entry specifies the base address and length of an area in memory where data should  
 55969 be placed. The *readv()* function shall always fill an area completely before proceeding to the  
 55970 next.

55971 Upon successful completion, *readv()* shall mark for update the last data access timestamp of the  
 55972 file.

55973 **RETURN VALUE**55974 Refer to *read()*.55975 **ERRORS**55976 Refer to *read()*.55977 In addition, the *readv()* function shall fail if:55978 [EINVAL] The sum of the *iov\_len* values in the *iov* array overflowed an **ssize\_t**.55979 The *readv()* function may fail if:55980 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.55981 **EXAMPLES**55982 **Reading Data into an Array**

55983 The following example reads data from the file associated with the file descriptor *fd* into the  
 55984 buffers specified by members of the *iov* array.

```
55985 #include <sys/types.h>
55986 #include <sys/uio.h>
55987 #include <unistd.h>
55988 ...
55989 ssize_t bytes_read;
55990 int fd;
55991 char buf0[20];
55992 char buf1[30];
55993 char buf2[40];
55994 int iovcnt;
55995 struct iovec iov[3];

55996 iov[0].iov_base = buf0;
55997 iov[0].iov_len = sizeof(buf0);
55998 iov[1].iov_base = buf1;
55999 iov[1].iov_len = sizeof(buf1);
56000 iov[2].iov_base = buf2;
```

```
56001     iov[2].iov_len = sizeof(buf2);
56002     ...
56003     iovcnt = sizeof(iov) / sizeof(struct iovec);
56004     bytes_read = readv(fd, iov, iovcnt);
56005     ...
```

56006 **APPLICATION USAGE**  
56007 None.

56008 **RATIONALE**  
56009 Refer to *read()*.

56010 **FUTURE DIRECTIONS**  
56011 None.

56012 **SEE ALSO**  
56013 *read()*, *writew()*  
56014 XBD <sys/uio.h>

56015 **CHANGE HISTORY**  
56016 First released in Issue 4, Version 2.

56017 **Issue 6**  
56018 Split out from the *read()* reference page.

56019 **Issue 7**  
56020 Changes are made related to support for finegrained timestamps.

**realloc()**56021 **NAME**56022            **realloc** — memory reallocator56023 **SYNOPSIS**

56024            #include &lt;stdlib.h&gt;

56025            void \*realloc(void \*ptr, size\_t size);

56026 **DESCRIPTION**

56027 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
 56028            conflict between the requirements described here and the ISO C standard is unintentional. This  
 56029            volume of POSIX.1-2008 defers to the ISO C standard.

56030            The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size  
 56031            specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new  
 56032            and old sizes. If the new size of the memory object would require movement of the object, the  
 56033            space for the previous instantiation of the object is freed. If the new size is larger, the contents of  
 56034            the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,  
 56035            the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

56036            If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

56037            If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has  
 56038            previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

56039            The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The  
 56040            pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 56041            a pointer to any type of object and then used to access such an object in the space allocated (until  
 56042            the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
 56043            disjoint from any other object. The pointer returned shall point to the start (lowest byte address)  
 56044            of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

56045 **RETURN VALUE**

56046            Upon successful completion with a size not equal to 0, *realloc()* shall return a pointer to the  
 56047            (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be  
 56048            successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*  
 56049 CX        shall return a null pointer and set *errno* to [ENOMEM].

56050 **ERRORS**

56051            The *realloc()* function shall fail if:

56052 CX        [ENOMEM]   Insufficient memory is available.

56053 **EXAMPLES**

56054            None.

56055 **APPLICATION USAGE**

56056            None.

56057 **RATIONALE**

56058            None.

56059 **FUTURE DIRECTIONS**

56060            None.

56061 **SEE ALSO**

56062            *calloc()*, *free()*, *malloc()*

56063            XBD <stdlib.h>

56064 **CHANGE HISTORY**

56065 First released in Issue 1. Derived from Issue 1 of the SVID.

56066 **Issue 6**

56067 Extensions beyond the ISO C standard are marked.

56068 The following new requirements on POSIX implementations derive from alignment with the  
56069 Single UNIX Specification:

- 56070 • In the RETURN VALUE section, if there is not enough available memory, the setting of  
56071 *errno* to [ENOMEM] is added.
- 56072 • The [ENOMEM] error condition is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**realpath()**56073 **NAME**56074 `realpath` — resolve a pathname56075 **SYNOPSIS**

```
56076 XSI #include <stdlib.h>
56077 char *realpath(const char *restrict file_name,
56078               char *restrict resolved_name);
```

56079 **DESCRIPTION**

56080 The `realpath()` function shall derive, from the pathname pointed to by `file_name`, an absolute  
 56081 pathname that resolves to the same directory entry, whose resolution does not involve '.',  
 56082 '..', or symbolic links. If `resolved_name` is a null pointer, the generated pathname shall be  
 56083 stored as a null-terminated string in a buffer allocated as if by a call to `malloc()`. Otherwise, if  
 56084 `{PATH_MAX}` is defined as a constant in the `<limits.h>` header, then the generated pathname  
 56085 shall be stored as a null-terminated string, up to a maximum of `{PATH_MAX}` bytes, in the  
 56086 buffer pointed to by `resolved_name`.

56087 If `resolved_name` is not a null pointer and `{PATH_MAX}` is not defined as a constant in the  
 56088 `<limits.h>` header, the behavior is undefined.

56089 **RETURN VALUE**

56090 Upon successful completion, `realpath()` shall return a pointer to the buffer containing the  
 56091 resolved name. Otherwise, `realpath()` shall return a null pointer and set `errno` to indicate the  
 56092 error.

56093 If the `resolved_name` argument is a null pointer, the pointer returned by `realpath()` can be passed  
 56094 to `free()`.

56095 If the `resolved_name` argument is not a null pointer and the `realpath()` function fails, the contents  
 56096 of the buffer pointed to by `resolved_name` are undefined.

56097 **ERRORS**

56098 The `realpath()` function shall fail if:

56099 [EACCES] Read or search permission was denied for a component of `file_name`.

56100 [EINVAL] The `file_name` argument is a null pointer.

56101 [EIO] An error occurred while reading from the file system.

56102 [ELOOP] A loop exists in symbolic links encountered during resolution of the `file_name`  
 56103 argument.

56104 [ENAMETOOLONG]

56105 The length of a component of a pathname is longer than `{NAME_MAX}`.

56106 [ENOENT] A component of `file_name` does not name an existing file or `file_name` points to  
 56107 an empty string.

56108 [ENOTDIR] A component of the path prefix is not a directory, or the `file_name` argument  
 56109 contains at least one non-`<slash>` character and ends with one or more trailing  
 56110 `<slash>` characters and the last pathname component names an existing file  
 56111 that is neither a directory nor a symbolic link to a directory.

56112 The `realpath()` function may fail if:

56113 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during  
 56114 resolution of the `file_name` argument.

- 56115 [ENAMETOOLONG]  
 56116 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 56117 symbolic link produced an intermediate result with a length that exceeds  
 56118 {PATH\_MAX}.
- 56119 [ENOMEM] Insufficient storage space is available.

56120 **EXAMPLES**56121 **Generating an Absolute Pathname**

56122 The following example generates an absolute pathname for the file identified by the *symlinkpath*  
 56123 argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
56124 #include <stdlib.h>
56125 ...
56126 char *symlinkpath = "/tmp/symlink/file";
56127 char *actualpath;

56128 actualpath = realpath(symlinkpath, NULL);
56129 if (actualpath != NULL)
56130 {
56131     ... use actualpath ...
56132     free(actualpath);
56133 }
56134 else
56135 {
56136     ... handle error ...
56137 }
```

56138 **APPLICATION USAGE**

56139 For functions that allocate memory as if by *malloc()*, the application should release such memory  
 56140 when it is no longer required by a call to *free()*. For *realpath()*, this is the return value.

56141 **RATIONALE**

56142 Since *realpath()* has no *length* argument, if {PATH\_MAX} is not defined as a constant in  
 56143 <limits.h>, applications have no way of determining how large a buffer they need to allocate for  
 56144 it to be safe to pass to *realpath()*. A {PATH\_MAX} value obtained from a prior *pathconf()* call is  
 56145 out-of-date by the time *realpath()* is called. Hence the only reliable way to use *realpath()* when  
 56146 {PATH\_MAX} is not defined in <limits.h> is to pass a null pointer for *resolved\_name* so that  
 56147 *realpath()* will allocate a buffer of the necessary size.

56148 **FUTURE DIRECTIONS**

56149 None.

56150 **SEE ALSO**

56151 *fpathconf()*, *free()*, *getcwd()*, *sysconf()*

56152 XBD <limits.h>, <stdlib.h>

56153 **CHANGE HISTORY**

56154 First released in Issue 4, Version 2.

56155 **Issue 5**

56156 Moved from X/OPEN UNIX extension to BASE.

**realpath()**56157 **Issue 6**

56158 The **restrict** keyword is added to the *realpath()* prototype for alignment with the  
56159 ISO/IEC 9899: 1999 standard.

56160 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
56161 [ELOOP] error condition is added.

56162 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the  
56163 DESCRIPTION for the case when *resolved\_name* is a null pointer, changing the [EINVAL] error  
56164 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

56165 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS  
56166 section to refer to the *file\_name* argument, rather than a nonexistent *path* argument.

56167 **Issue 7**

56168 Austin Group Interpretation 1003.1-2001 #143 is applied.

56169 This function is updated for passing a null pointer to *realpath()* for the *resolved\_name* argument.

56170 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
56171 *malloc()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

56172 **NAME**

56173           recv — receive a message from a connected socket

56174 **SYNOPSIS**

56175           #include &lt;sys/socket.h&gt;

56176           ssize\_t recv(int socket, void \*buffer, size\_t length, int flags);

56177 **DESCRIPTION**

56178           The *recv()* function shall receive a message from a connection-mode or connectionless-mode  
 56179           socket. It is normally used with connected sockets because it does not permit the application to  
 56180           retrieve the source address of received data.

56181           The *recv()* function takes the following arguments:56182           *socket*           Specifies the socket file descriptor.56183           *buffer*           Points to a buffer where the message should be stored.56184           *length*           Specifies the length in bytes of the buffer pointed to by the *buffer* argument.56185           *flags*           Specifies the type of message reception. Values of this argument are formed by  
 56186           logically OR'ing zero or more of the following values:56187           MSG\_PEEK           Peeks at an incoming message. The data is treated as unread and  
 56188           the next *recv()* or similar function shall still return this data.56189           MSG\_OOB           Requests out-of-band data. The significance and semantics of  
 56190           out-of-band data are protocol-specific.56191           MSG\_WAITALL       On SOCK\_STREAM sockets this requests that the function block  
 56192           until the full amount of data can be returned. The function may  
 56193           return the smaller amount of data if the socket is a message-  
 56194           based socket, if a signal is caught, if the connection is terminated,  
 56195           if MSG\_PEEK was specified, or if an error is pending for the  
 56196           socket.

56197           The *recv()* function shall return the length of the message written to the buffer pointed to by the  
 56198           *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET,  
 56199           the entire message shall be read in a single operation. If a message is too long to fit in the  
 56200           supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess bytes shall be  
 56201           discarded. For stream-based sockets, such as SOCK\_STREAM, message boundaries shall be  
 56202           ignored. In this case, data shall be returned to the user as soon as it becomes available, and no  
 56203           data shall be discarded.

56204           If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 56205           message.56206           If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 56207           descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket  
 56208           and O\_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to  
 56209           [EAGAIN] or [EWOULDBLOCK].56210 **RETURN VALUE**56211           Upon successful completion, *recv()* shall return the length of the message in bytes. If no  
 56212           messages are available to be received and the peer has performed an orderly shutdown, *recv()*  
 56213           shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

**recv()**56214 **ERRORS**56215 The *recv()* function shall fail if:

56216 [EAGAIN] or [EWOULDBLOCK]

56217 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
56218 to be received; or MSG\_OOB is set and no out-of-band data is available and  
56219 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
56220 not support blocking to await out-of-band data.56221 [EBADF] The *socket* argument is not a valid file descriptor.

56222 [ECONNRESET] A connection was forcibly closed by a peer.

56223 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any  
56224 data was available.

56225 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.

56226 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

56227 [ENOTSOCK] The *socket* argument does not refer to a socket.

56228 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

56229 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
56230 transmission timeout on active connection.56231 The *recv()* function may fail if:

56232 [EIO] An I/O error occurred while reading from or writing to the file system.

56233 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

56234 [ENOMEM] Insufficient memory was available to fulfill the request.

56235 **EXAMPLES**

56236 None.

56237 **APPLICATION USAGE**56238 The *recv()* function is equivalent to *recvfrom()* with a zero *address\_len* argument, and to *read()* if  
56239 no flags are used.56240 The *select()* and *poll()* functions can be used to determine when data is available to be received.56241 **RATIONALE**

56242 None.

56243 **FUTURE DIRECTIONS**

56244 None.

56245 **SEE ALSO**56246 *poll()*, *pselect()*, *read()*, *recvmsg()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,  
56247 *write()*

56248 XBD &lt;sys/socket.h&gt;

56249 **CHANGE HISTORY**

56250 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

56251 **NAME**

56252 recvfrom — receive a message from a socket

56253 **SYNOPSIS**

56254 #include &lt;sys/socket.h&gt;

```
56255 ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
56256                 int flags, struct sockaddr *restrict address,
56257                 socklen_t *restrict address_len);
```

56258 **DESCRIPTION**

56259 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode  
 56260 socket. It is normally used with connectionless-mode sockets because it permits the application  
 56261 to retrieve the source address of received data.

56262 The *recvfrom()* function takes the following arguments:

56263	<i>socket</i>	Specifies the socket file descriptor.
56264	<i>buffer</i>	Points to the buffer where the message should be stored.
56265	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
56266	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
56268	MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next <i>recvfrom()</i> or similar function shall still return this data.
56269		
56271	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
56272		
56273	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
56274		
56275		
56276		
56277		
56278		
56279	<i>address</i>	A null pointer, or points to a <b>sockaddr</b> structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.
56280		
56281		
56282	<i>address_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i> argument.
56283		

56284 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by  
 56285 RS the *buffer* argument. For message-based sockets, such as SOCK\_RAW, SOCK\_DGRAM, and  
 56286 SOCK\_SEQPACKET, the entire message shall be read in a single operation. If a message is too  
 56287 long to fit in the supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess  
 56288 bytes shall be discarded. For stream-based sockets, such as SOCK\_STREAM, message  
 56289 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes  
 56290 available, and no data shall be discarded.

56291 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 56292 message.

56293 Not all protocols provide the source address for messages. If the *address* argument is not a null  
 56294 pointer and the protocol provides the source address of messages, the source address of the

**recvfrom()**

56295 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,  
 56296 and the length of this address shall be stored in the object pointed to by the *address\_len*  
 56297 argument.

56298 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
 56299 the stored address shall be truncated.

56300 If the *address* argument is not a null pointer and the protocol does not provide the source address  
 56301 of messages, the value stored in the object pointed to by *address* is unspecified.

56302 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 56303 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the  
 56304 socket and O\_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*  
 56305 to [EAGAIN] or [EWOULDBLOCK].

**RETURN VALUE**

56306 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no  
 56307 messages are available to be received and the peer has performed an orderly shutdown,  
 56308 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the  
 56309 error.  
 56310

**ERRORS**

56311 The *recvfrom()* function shall fail if:

56312 [EAGAIN] or [EWOULDBLOCK]

56313 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 56314 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 56315 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 56316 not support blocking to await out-of-band data.  
 56317

56318 [EBADF] The *socket* argument is not a valid file descriptor.

56319 [ECONNRESET] A connection was forcibly closed by a peer.

56320 [EINTR] A signal interrupted *recvfrom()* before any data was available.

56321 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.

56322 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

56323 [ENOTSOCK] The *socket* argument does not refer to a socket.

56324 [EOPNOTSUPP] The specified flags are not supported for this socket type.

56325 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 56326 transmission timeout on active connection.

56327 The *recvfrom()* function may fail if:

56328 [EIO] An I/O error occurred while reading from or writing to the file system.

56329 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

56330 [ENOMEM] Insufficient memory was available to fulfill the request.

56331 **EXAMPLES**

56332 None.

56333 **APPLICATION USAGE**56334 The *select()* and *poll()* functions can be used to determine when data is available to be received.56335 **RATIONALE**

56336 None.

56337 **FUTURE DIRECTIONS**

56338 None.

56339 **SEE ALSO**56340 *poll()*, *pselect()*, *read()*, *recv()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write()*56341 XBD <[sys/socket.h](#)>56342 **CHANGE HISTORY**

56343 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**recvmsg()**56344 **NAME**56345 `recvmsg` — receive a message from a socket56346 **SYNOPSIS**56347 `#include <sys/socket.h>`56348 `ssize_t recvmsg(int socket, struct msghdr *message, int flags);`56349 **DESCRIPTION**56350 The `recvmsg()` function shall receive a message from a connection-mode or connectionless-mode  
56351 socket. It is normally used with connectionless-mode sockets because it permits the application  
56352 to retrieve the source address of received data.56353 The `recvmsg()` function takes the following arguments:

56354	<i>socket</i>	Specifies the socket file descriptor.
56355	<i>message</i>	Points to a <b>msghdr</b> structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
56356		
56357		
56358		
56359	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
56360		
56361	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
56362		
56363	MSG_PEEK	Peeks at the incoming message.
56364	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
56365		
56366		
56367		
56368		
56369		

56370 The `recvmsg()` function shall receive messages from unconnected or connected sockets and shall  
56371 return the length of the message.56372 The `recvmsg()` function shall return the total length of the message. For message-based sockets,  
56373 such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single  
56374 operation. If a message is too long to fit in the supplied buffers, and MSG\_PEEK is not set in the  
56375 *flags* argument, the excess bytes shall be discarded, and MSG\_TRUNC shall be set in the  
56376 *msg\_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK\_STREAM,  
56377 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it  
56378 becomes available, and no data shall be discarded.56379 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
56380 message.56381 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
56382 descriptor, `recvmsg()` shall block until a message arrives. If no messages are available at the  
56383 socket and O\_NONBLOCK is set on the socket's file descriptor, the `recvmsg()` function shall fail  
56384 and set *errno* to [EAGAIN] or [EWOULDBLOCK].56385 In the **msghdr** structure, the *msg\_name* and *msg\_namelen* members specify the source address if  
56386 the socket is unconnected. If the socket is connected, the *msg\_name* and *msg\_namelen* members  
56387 shall be ignored. The *msg\_name* member may be a null pointer if no names are desired or  
56388 required. The *msg\_iov* and *msg\_iovlen* fields are used to specify where the received data shall be

56389 stored. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 56390 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 56391 gives its size in bytes. Each storage area indicated by *msg\_iov* is filled with received data in turn  
 56392 until all of the received data is stored or all of the areas have been filled.

56393 Upon successful completion, the *msg\_flags* member of the message header shall be the bitwise-  
 56394 inclusive OR of all of the following flags that indicate conditions detected for the received  
 56395 message:

56396 MSG\_EOR End-of-record was received (if supported by the protocol).

56397 MSG\_OOB Out-of-band data was received.

56398 MSG\_TRUNC Normal data was truncated.

56399 MSG\_CTRUNC Control data was truncated.

#### 56400 RETURN VALUE

56401 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no  
 56402 messages are available to be received and the peer has performed an orderly shutdown,  
 56403 *recvmsg()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

#### 56404 ERRORS

56405 The *recvmsg()* function shall fail if:

56406 [EAGAIN] or [EWOULDBLOCK]

56407 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 56408 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 56409 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 56410 not support blocking to await out-of-band data.

56411 [EBADF] The *socket* argument is not a valid open file descriptor.

56412 [ECONNRESET] A connection was forcibly closed by a peer.

56413 [EINTR] This function was interrupted by a signal before any data was available.

56414 [EINVAL] The sum of the *iov\_len* values overflows a **ssize\_t**, or the MSG\_OOB flag is set  
 56415 and no out-of-band data is available.

56416 [EMSGSIZE] The *msg\_iovlen* member of the **msghdr** structure pointed to by *message* is less  
 56417 than or equal to 0, or is greater than {IOV\_MAX}.

56418 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

56419 [ENOTSOCK] The *socket* argument does not refer to a socket.

56420 [EOPNOTSUPP] The specified flags are not supported for this socket type.

56421 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 56422 transmission timeout on active connection.

56423 The *recvmsg()* function may fail if:

56424 [EIO] An I/O error occurred while reading from or writing to the file system.

56425 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

56426 [ENOMEM] Insufficient memory was available to fulfill the request.

**recvmsg()**

System Interfaces

56427 **EXAMPLES**

56428 None.

56429 **APPLICATION USAGE**56430 The *select()* and *poll()* functions can be used to determine when data is available to be received.56431 **RATIONALE**

56432 None.

56433 **FUTURE DIRECTIONS**

56434 None.

56435 **SEE ALSO**56436 *poll()*, *pselect()*, *recv()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*56437 XBD [<sys/socket.h>](#)56438 **CHANGE HISTORY**

56439 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 56440 NAME

56441 regcomp, regerror, regex, regfree — regular expression matching

## 56442 SYNOPSIS

```
56443 #include <regex.h>
56444 int regcomp(regex_t *restrict preg, const char *restrict pattern,
56445             int cflags);
56446 size_t regerror(int errcode, const regex_t *restrict preg,
56447               char *restrict errbuf, size_t errbuf_size);
56448 int regex(const regex_t *restrict preg, const char *restrict string,
56449           size_t nmatch, regmatch_t pmatch[restrict], int eflags);
56450 void regfree(regex_t *preg);
```

## 56451 DESCRIPTION

56452 These functions interpret *basic* and *extended* regular expressions as described in XBD [Chapter 9](#)  
 56453 (on page 181).

56454 The **regex\_t** structure is defined in **<regex.h>** and contains at least the following member:

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

56458 The **regmatch\_t** structure is defined in **<regex.h>** and contains at least the following members:

Member Type	Member Name	Description
regoff_t	rm_so	Byte offset from start of <i>string</i> to start of substring.
regoff_t	rm_eo	Byte offset from start of <i>string</i> of the first character after the end of substring.

56464 The *regcomp()* function shall compile the regular expression contained in the string pointed to by  
 56465 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*  
 56466 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in  
 56467 the **<regex.h>** header:

- 56468 REG\_EXTENDED Use Extended Regular Expressions.
- 56469 REG\_ICASE Ignore case in match (see XBD [Chapter 9](#), on page 181).
- 56470 REG\_NOSUB Report only success/fail in *regex()*.
- 56471 REG\_NEWLINE Change the handling of <newline> characters, as described in the text.

56472 The default regular expression type for *pattern* is a Basic Regular Expression. The application can  
 56473 specify Extended Regular Expressions using the REG\_EXTENDED *cflags* flag.

56474 If the REG\_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re\_nsub* to the number of  
 56475 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in  
 56476 extended regular expressions) found in *pattern*.

56477 The *regex()* function compares the null-terminated string specified by *string* with the compiled  
 56478 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regex()*  
 56479 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The  
 56480 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are  
 56481 defined in the **<regex.h>** header:

## regcomp()

- 56482 REG\_NOTBOL The first character of the string pointed to by *string* is not the beginning of  
56483 the line. Therefore, the <circumflex> character ('^'), when taken as a  
56484 special character, shall not match the beginning of *string*.
- 56485 REG\_NOTEOL The last character of the string pointed to by *string* is not the end of the  
56486 line. Therefore, the <dollar-sign> ('\$'), when taken as a special character,  
56487 shall not match the end of *string*.
- 56488 If *nmatch* is 0 or REG\_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall  
56489 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument  
56490 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that  
56491 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions  
56492 of *pattern*: *pmatch[i].rm\_so* shall be the byte offset of the beginning and *pmatch[i].rm\_eo* shall be  
56493 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th  
56494 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that  
56495 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*  
56496 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself  
56497 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first  
56498 *nmatch* substrings.
- 56499 When matching a basic or extended regular expression, any given parenthesized subexpression  
56500 of *pattern* might participate in the match of several different substrings of *string*, or it might not  
56501 match any substring even though the pattern as a whole did match. The following rules shall be  
56502 used to determine which substrings to report in *pmatch* when matching regular expressions:
- 56503 1. If subexpression *i* in a regular expression is not contained within another subexpression,  
56504 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall  
56505 delimit the last such match.
  - 56506 2. If subexpression *i* is not contained within another subexpression, and it did not  
56507 participate in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A  
56508 subexpression does not participate in the match when:
    - 56509 ' \* ' or "\{\}" appears immediately after the subexpression in a basic regular  
56510 expression, or ' \* ', ' ? ', or "{ }" appears immediately after the subexpression in  
56511 an extended regular expression, and the subexpression did not match (matched 0  
56512 times)
- 56513 or:
- 56514 ' | ' is used in an extended regular expression to select this subexpression or  
56515 another, and the other subexpression matched.
- 56516 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained  
56517 within any other subexpression that is contained within *j*, and a match of subexpression *j*  
56518 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in  
56519 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in  
56520 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start  
56521 of *string*.
  - 56522 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,  
56523 then the pointers in *pmatch[i]* shall also be -1.
  - 56524 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be  
56525 the byte offset of the character or null terminator immediately following the zero-length  
56526 string.

56527 If, when *regexec()* is called, the locale is different from when the regular expression was  
56528 compiled, the result is undefined.

56529 If REG\_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an  
56530 ordinary character. If REG\_NEWLINE is set, then <newline> shall be treated as an ordinary  
56531 character except as follows:

- 56532 1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or  
56533 by any form of a non-matching list (see XBD Chapter 9, on page 181).
- 56534 2. A <circumflex> (' ^ ') in *pattern*, when used to specify expression anchoring (see XBD  
56535 Section 9.3.8, on page 187), shall match the zero-length string immediately after a  
56536 <newline> in *string*, regardless of the setting of REG\_NOTBOL.
- 56537 3. A <dollar-sign> (' \$ ') in *pattern*, when used to specify expression anchoring, shall match  
56538 the zero-length string immediately before a <newline> in *string*, regardless of the setting  
56539 of REG\_NOTEOL.

56540 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

56541 The following constants are defined as error return values:

56542	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than 56543 two numbers, first larger than second.
56544	REG_BADPAT	Invalid regular expression.
56545	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
56546	REG_EBRACE	"\{\}" imbalance.
56547	REG_EBRACK	"[]" imbalance.
56548	REG_ECOLLATE	Invalid collating element referenced.
56549	REG_ECTYPE	Invalid character class type referenced.
56550	REG_EESCAPE	Trailing <backslash> character in pattern.
56551	REG_EPAREN	"(\)" or "()" imbalance.
56552	REG_ERANGE	Invalid endpoint in range expression.
56553	REG_ESPACE	Out of memory.
56554	REG_ESUBREG	Number in "\digit" invalid or in error.
56555	REG_NOMATCH	<i>regexec()</i> failed to match.

56556 If more than one error occurs in processing a function call, any one of the possible constants may  
56557 be returned, as the order of detection is unspecified.

56558 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and  
56559 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the  
56560 *errcode* argument, which the application shall ensure is the last non-zero value returned by  
56561 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of  
56562 the generated string is unspecified.

56563 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,  
56564 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not  
56565 be as detailed under some implementations.

56566 If the *errbuf\_size* argument is not 0, *regerror()* shall place the generated string into the buffer of

**regcomp()**

56567 size *errbuf\_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit  
56568 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

56569 If *errbuf\_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer  
56570 needed to hold the generated string.

56571 If the *preg* argument to *regexexec()* or *regfree()* is not a compiled regular expression returned by  
56572 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression  
56573 after it is given to *regfree()*.

**RETURN VALUE**

56574 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an  
56575 integer value indicating an error as described in **<regex.h>**, and the content of *preg* is undefined.  
56576 If a code is returned, the interpretation shall be as given in **<regex.h>**.  
56577

56578 If *regcomp()* detects an invalid RE, it may return REG\_BADPAT, or it may return one of the error  
56579 codes that more precisely describes the error.

56580 Upon successful completion, the *regexexec()* function shall return 0. Otherwise, it shall return  
56581 REG\_NOMATCH to indicate no match.

56582 Upon successful completion, the *regerror()* function shall return the number of bytes needed to  
56583 hold the entire generated string, including the null termination. If the return value is greater  
56584 than *errbuf\_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

56585 The *regfree()* function shall not return a value.

**ERRORS**

56586 No errors are defined.  
56587

**EXAMPLES**

```
56588 #include <regex.h>
56589
56590 /*
56591  * Match string against the extended regular expression in
56592  * pattern, treating errors as no match.
56593  *
56594  * Return 1 for match, 0 for no match.
56595  */
56596
56597 int
56598 match(const char *string, char *pattern)
56599 {
56600     int status;
56601     regex_t re;
56602
56603     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
56604         return(0); /* Report error. */
56605     }
56606     status = regexexec(&re, string, (size_t) 0, NULL, 0);
56607     regfree(&re);
56608     if (status != 0) {
56609         return(0); /* Report error. */
56610     }
56611     return(1);
56612 }
```

56611 The following demonstrates how the REG\_NOTBOL flag could be used with *regexexec()* to find all

56612 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very  
56613 little error checking is done.)

```
56614 (void) regcomp (&re, pattern, 0);
56615 /* This call to regexec() finds the first match on the line. */
56616 error = regexec (&re, &buffer[0], 1, &pm, 0);
56617 while (error == 0) { /* While matches found. */
56618     /* Substring found between pm.rm_so and pm.rm_eo. */
56619     /* This call to regexec() finds the next match. */
56620     error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
56621 }
```

#### 56622 APPLICATION USAGE

56623 An application could use:

```
56624 regerror(code, preg, (char *)NULL, (size_t)0)
```

56625 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the  
56626 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,  
56627 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger  
56628 buffer if it finds that this is too small.

56629 To match a pattern as described in XCU Section 2.13 (on page 2332), use the *fnmatch()* function.

#### 56630 RATIONALE

56631 The *regexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are  
56632 supplied by the application, even if some elements of *pmatch* do not correspond to  
56633 subexpressions in *pattern*. The application developer should note that there is probably no  
56634 reason for using a value of *nmatch* that is larger than *preg->re\_nsub+1*.

56635 The REG\_NEWLINE flag supports a use of RE matching that is needed in some applications like  
56636 text editors. In such applications, the user supplies an RE asking the application to find a line  
56637 that matches the given expression. An anchor in such an RE anchors at the beginning or end of  
56638 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a  
56639 single long string and specify REG\_NEWLINE to *regcomp()* to get the desired behavior. The  
56640 application must ensure that there are no explicit <newline> characters in *pattern* if it wants to  
56641 ensure that any match occurs entirely within a single line.

56642 The REG\_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to  
56643 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate  
56644 the same compiled RE in *regcomp()* regardless of the setting of REG\_NEWLINE and have  
56645 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will  
56646 generate different compiled REs based on the REG\_NEWLINE.

56647 The REG\_ICASE flag supports the operations taken by the *grep -i* option and the historical  
56648 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be  
56649 written that does the same thing as these utilities.

56650 The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather  
56651 than pointers. This allows type-safe access to both constant and non-constant strings.

56652 The type **regoff\_t** is used for the elements of *pmatch[]* to ensure that the application can  
56653 represent large arrays in memory (important for an application conforming to the Shell and  
56654 Utilities volume of POSIX.1-2008).

56655 The 1992 edition of this standard required **regoff\_t** to be at least as wide as **off\_t**, to facilitate  
56656 future extensions in which the string to be searched is taken from a file. However, these future  
56657 extensions have not appeared. The requirement rules out popular implementations with 32-bit

**regcomp()**

- 56658 **regoff\_t** and 64-bit **off\_t**, so it has been removed.
- 56659 The standard developers rejected the inclusion of a *regsub()* function that would be used to do  
56660 substitutions for a matched RE. While such a routine would be useful to some applications, its  
56661 utility would be much more limited than the matching function described here. Both RE parsing  
56662 and substitution are possible to implement without support other than that required by the  
56663 ISO C standard, but matching is much more complex than substituting. The only difficult part of  
56664 substitution, given the information supplied by *regexexec()*, is finding the next character in a string  
56665 when there can be multi-byte characters. That is a much larger issue, and one that needs a more  
56666 general solution.
- 56667 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for  
56668 this feature.
- 56669 The interface is defined so that the matched substrings *rm\_sp* and *rm\_ep* are in a separate  
56670 **regmatch\_t** structure instead of in **regex\_t**. This allows a single compiled RE to be used  
56671 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple  
56672 threads of lightweight processes. (The *preg* argument to *regexexec()* is declared with type **const**, so  
56673 the implementation is not permitted to use the structure to store intermediate results.) It also  
56674 allows an application to request an arbitrary number of substrings from an RE. The number of  
56675 subexpressions in the RE is reported in *re\_nsub* in *preg*. With this change to *regexexec()*,  
56676 consideration was given to dropping the REG\_NOSUB flag since the user can now specify this  
56677 with a zero *nmatch* argument to *regexexec()*. However, keeping REG\_NOSUB allows an  
56678 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()* that  
56679 no subexpressions need be reported. The implementation is only required to fill in *pmatch* if  
56680 *nmatch* is not zero and if REG\_NOSUB is not specified. Note that the **size\_t** type, as defined in  
56681 the ISO C standard, is unsigned, so the description of *regexexec()* does not need to address  
56682 negative values of *nmatch*.
- 56683 REG\_NOTBOL was added to allow an application to do repeated searches for the same pattern  
56684 in a line. If the pattern contains a <circumflex> character that should match the beginning of a  
56685 line, then the pattern should only match when matched against the beginning of the line.  
56686 Without the REG\_NOTBOL flag, the application could rewrite the expression for subsequent  
56687 matches, but in the general case this would require parsing the expression. The need for  
56688 REG\_NOTEOL is not as clear; it was added for symmetry.
- 56689 The addition of the *regerror()* function addresses the historical need for conforming application  
56690 programs to have access to error information more than "Function failed to compile/match your  
56691 RE for unknown reasons".
- 56692 This interface provides for two different methods of dealing with error conditions. The specific  
56693 error codes (REG\_EBRACE, for example), defined in <**regex.h**>, allow an application to recover  
56694 from an error if it is so able. Many applications, especially those that use patterns supplied by a  
56695 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-  
56696 readable error message to present to the user.
- 56697 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating  
56698 memory to hold the generated string. The scheme used by *sterror()* in the ISO C standard was  
56699 considered unacceptable since it creates difficulties for multi-threaded applications.
- 56700 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more  
56701 descriptive message than would be possible with *errcode* alone. An implementation might, for  
56702 example, save the character offset of the offending character of the pattern in a field of *preg*, and  
56703 then include that in the generated message string. The implementation may also ignore *preg*.
- 56704 A REG\_FILENAME flag was considered, but omitted. This flag caused *regexexec()* to match

56705 patterns as described in XCU Section 2.13 (on page 2332) instead of REs. This service is now  
56706 provided by the *fnmatch()* function.

56707 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and  
56708 POSIX.1-2008 in how to handle a “bad” regular expression. The ISO POSIX-2:1993 standard says  
56709 that many bad constructs “produce undefined results”, or that “the interpretation is undefined”.  
56710 POSIX.1-2008, however, says that the interpretation of such REs is unspecified. The term  
56711 “undefined” means that the action by the application is an error, of similar severity to passing a  
56712 bad pointer to a function.

56713 The *regcomp()* and *regexec()* functions are required to accept any null-terminated string as the  
56714 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is  
56715 “unspecified”. POSIX.1-2008 does not specify how the functions will interpret the pattern; they  
56716 might return error codes, or they might do pattern matching in some completely unexpected  
56717 way, but they should not do something like abort the process.

#### 56718 FUTURE DIRECTIONS

56719 None.

#### 56720 SEE ALSO

56721 *fnmatch()*, *glob()*

56722 XBD Chapter 9 (on page 181), `<regex.h>`, `<sys/types.h>`

56723 XCU Section 2.13 (on page 2332)

#### 56724 CHANGE HISTORY

56725 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 56726 Issue 5

56727 Moved from POSIX2 C-language Binding to BASE.

#### 56728 Issue 6

56729 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

56730 The following new requirements on POSIX implementations derive from alignment with the  
56731 Single UNIX Specification:

- 56732 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
56733 required for conforming implementations of previous POSIX specifications, it was not  
56734 required for UNIX applications.

56735 The normative text is updated to avoid use of the term “must” for application requirements.

56736 The `REG_ENOSYS` constant is removed.

56737 The `restrict` keyword is added to the *regcomp()*, *regerror()*, and *regexec()* prototypes for  
56738 alignment with the ISO/IEC 9899:1999 standard.

#### 56739 Issue 7

56740 Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error  
56741 occurs in processing a function call, any one of the possible constants may be returned.

56742 SD5-XBD-ERN-60 is applied.

**remainder()**56743 **NAME**

56744 remainder, remainderf, remainderl — remainder function

56745 **SYNOPSIS**

56746 #include &lt;math.h&gt;

56747 double remainder(double x, double y);

56748 float remainderf(float x, float y);

56749 long double remainderl(long double x, long double y);

56750 **DESCRIPTION**56751 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
56752 conflict between the requirements described here and the ISO C standard is unintentional. This  
56753 volume of POSIX.1-2008 defers to the ISO C standard.56754 These functions shall return the floating-point remainder  $r=x-ny$  when  $y$  is non-zero. The value  
56755  $n$  is the integral value nearest the exact value  $x/y$ . When  $|n-x/y| = 1/2$ , the value  $n$  is chosen to  
56756 be even.56757 The behavior of *remainder()* shall be independent of the rounding mode.56758 **RETURN VALUE**56759 Upon successful completion, these functions shall return the floating-point remainder  $r=x-ny$   
56760 when  $y$  is non-zero.56761 On systems that do not support the IEC 60559 Floating-Point option, if  $y$  is zero, it is  
56762 implementation-defined whether a domain error occurs or zero is returned.56763 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.56764 If  $x$  is infinite or  $y$  is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if  
56765 supported), or an implementation-defined value shall be returned.56766 **ERRORS**

56767 These functions shall fail if:

56768 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is non-  
56769 NaN.56770 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
56771 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
56772 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
56773 shall be raised.

56774 These functions may fail if:

56775 **Domain Error** The  $y$  argument is zero.56776 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
56777 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
56778 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
56779 shall be raised.

56780 **EXAMPLES**

56781 None.

56782 **APPLICATION USAGE**56783 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
56784 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.56785 **RATIONALE**

56786 None.

56787 **FUTURE DIRECTIONS**

56788 None.

56789 **SEE ALSO**56790 *abs()*, *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*

56791 XBD Section 4.19 (on page 116), &lt;math.h&gt;

56792 **CHANGE HISTORY**

56793 First released in Issue 4, Version 2.

56794 **Issue 5**

56795 Moved from X/OPEN UNIX extension to BASE.

56796 **Issue 6**56797 The *remainder()* function is no longer marked as an extension.56798 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999  
56799 standard.56800 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
56801 revised to align with the ISO/IEC 9899:1999 standard.56802 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
56803 marked.56804 **Issue 7**

56805 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

**remove()**56806 **NAME**56807 `remove` — remove a file56808 **SYNOPSIS**56809 `#include <stdio.h>`56810 `int remove(const char *path);`56811 **DESCRIPTION**56812 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
56813 conflict between the requirements described here and the ISO C standard is unintentional. This  
56814 volume of POSIX.1-2008 defers to the ISO C standard.56815 The `remove()` function shall cause the file named by the pathname pointed to by `path` to be no  
56816 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,  
56817 unless it is created anew.56818 CX If `path` does not name a directory, `remove(path)` shall be equivalent to `unlink(path)`.56819 If `path` names a directory, `remove(path)` shall be equivalent to `rmdir(path)`.56820 **RETURN VALUE**56821 CX Refer to `rmdir()` or `unlink()`.56822 **ERRORS**56823 CX Refer to `rmdir()` or `unlink()`.56824 **EXAMPLES**56825 **Removing Access to a File**56826 The following example shows how to remove access to a file named `/home/cnd/old_mods`.56827 `#include <stdio.h>`  
56828 `int status;`  
56829 `...`  
56830 `status = remove("/home/cnd/old_mods");`56831 **APPLICATION USAGE**

56832 None.

56833 **RATIONALE**

56834 None.

56835 **FUTURE DIRECTIONS**

56836 None.

56837 **SEE ALSO**56838 `rmdir()`, `unlink()`56839 XBD `<stdio.h>`56840 **CHANGE HISTORY**56841 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C  
56842 standard.56843 **Issue 6**

56844 Extensions beyond the ISO C standard are marked.

56845 The following new requirements on POSIX implementations derive from alignment with the  
56846 Single UNIX Specification:

- 56847
- 56848 • The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is  
56849 not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to  
*rmdir()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**remque()***System Interfaces*56850 **NAME**

56851 remque — remove an element from a queue

56852 **SYNOPSIS**56853 XSI `#include <search.h>`56854 `void remque(void *element);`56855 **DESCRIPTION**56856 Refer to *insque()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

56857 **NAME**

56858 remquo, remquof, remquol — remainder functions

56859 **SYNOPSIS**

56860 #include &lt;math.h&gt;

56861 double remquo(double x, double y, int \*quo);

56862 float remquof(float x, float y, int \*quo);

56863 long double remquol(long double x, long double y, int \*quo);

56864 **DESCRIPTION**

56865 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 56866 conflict between the requirements described here and the ISO C standard is unintentional. This  
 56867 volume of POSIX.1-2008 defers to the ISO C standard.

56868 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the  
 56869 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by *quo*,  
 56870 they store a value whose sign is the sign of  $x/y$  and whose magnitude is congruent modulo  $2^n$   
 56871 to the magnitude of the integral quotient of  $x/y$ , where  $n$  is an implementation-defined integer  
 56872 greater than or equal to 3. If  $y$  is zero, the value stored in the object pointed to by *quo* is  
 56873 unspecified.

56874 An application wishing to check for error situations should set *errno* to zero and call  
 56875 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 56876 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 56877 zero, an error has occurred.

56878 **RETURN VALUE**56879 These functions shall return  $x \text{ REM } y$ .

56880 On systems that do not support the IEC 60559 Floating-Point option, if  $y$  is zero, it is  
 56881 implementation-defined whether a domain error occurs or zero is returned.

56882 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

56883 If  $x$  is  $\pm\text{Inf}$  or  $y$  is zero and the other argument is non-NaN, a domain error shall occur, and either  
 56884 a NaN (if supported), or an implementation-defined value shall be returned.

56885 **ERRORS**

56886 These functions shall fail if:

56887 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is non-  
 56888 NaN.

56889 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 56890 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 56891 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 56892 shall be raised.

56893 These functions may fail if:

56894 **Domain Error** The  $y$  argument is zero.

56895 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 56896 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 56897 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 56898 shall be raised.

**remquo()**56899 **EXAMPLES**

56900 None.

56901 **APPLICATION USAGE**56902 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
56903 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.56904 **RATIONALE**56905 These functions are intended for implementing argument reductions which can exploit a few  
56906 low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an  
56907 exact representation of the quotient is not practical.56908 **FUTURE DIRECTIONS**

56909 None.

56910 **SEE ALSO**56911 *feclearexcept()*, *fetestexcept()*, *remainder()*

56912 XBD Section 4.19 (on page 116), &lt;math.h&gt;

56913 **CHANGE HISTORY**

56914 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

56915 **Issue 7**

56916 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

56917 **NAME**

56918 rename, renameat — rename file relative to directory file descriptor

56919 **SYNOPSIS**

56920 #include &lt;stdio.h&gt;

56921 int rename(const char \*old, const char \*new);

56922 CX int renameat(int oldfd, const char \*old, int newfd,  
56923 const char \*new);56924 **DESCRIPTION**56925 CX For *rename()*: The functionality described on this reference page is aligned with the ISO C  
56926 standard. Any conflict between the requirements described here and the ISO C standard is  
56927 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.56928 The *rename()* function shall change the name of a file. The *old* argument points to the pathname  
56929 CX of the file to be renamed. The *new* argument points to the new pathname of the file. If the *new*  
56930 argument does not resolve to an existing directory entry for a file of type directory and the *new*  
56931 argument contains at least one non-*</i>slash*>* character and ends with one or more trailing  
56932 *</i>slash*>* characters after all symbolic links have been processed, *rename()* shall fail.**56933 If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic  
56934 link itself, and shall not resolve the last component of the argument. If the *old* argument and the  
56935 *new* argument resolve to either the same existing directory entry or different directory entries for  
56936 the same existing file, *rename()* shall return successfully and perform no other action.56937 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall  
56938 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be  
56939 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other  
56940 processes throughout the renaming operation and refer either to the file referred to by *new* or *old*  
56941 before the operation began. Write access permission is required for both the directory containing  
56942 *old* and the directory containing *new*.56943 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the  
56944 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it  
56945 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout  
56946 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the  
56947 operation began. If *new* names an existing directory, it shall be required to be an empty directory.56948 If either *pathname* argument refers to a path whose final component is either dot or dot-dot,  
56949 *rename()* shall fail.56950 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.  
56951 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.56952 The *old* pathname shall not name an ancestor directory of the *new* pathname. Write access  
56953 permission is required for the directory containing *old* and the directory containing *new*. If the  
56954 *old* argument points to the pathname of a directory, write access permission may be required for  
56955 the directory named by *old*, and, if it exists, the directory named by *new*.56956 If the link named by the *new* argument exists and the file's link count becomes 0 when it is  
56957 removed and no process has the file open, the space occupied by the file shall be freed and the  
56958 file shall no longer be accessible. If one or more processes have the file open when the last link is  
56959 removed, the link shall be removed before *rename()* returns, but the removal of the file contents  
56960 shall be postponed until all references to the file are closed.56961 Upon successful completion, *rename()* shall mark for update the last data modification and last

**rename()**

56962 file status change timestamps of the parent directory of each file.

56963 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be  
56964 unaffected.

56965 The *renameat()* function shall be equivalent to the *rename()* function except in the case where  
56966 either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located  
56967 relative to the directory associated with the file descriptor *oldfd* instead of the current working  
56968 directory. If *new* is a relative path, the same happens only relative to the directory associated  
56969 with *newfd*. If the file descriptor was opened without O\_SEARCH, the function shall check  
56970 whether directory searches are permitted using the current permissions of the directory  
56971 underlying the file descriptor. If the file descriptor was opened with O\_SEARCH, the function  
56972 shall not perform the check.

56973 If *renameat()* is passed the special value AT\_FDCWD in the *oldfd* or *newfd* parameter, the current  
56974 working directory shall be used in the determination of the file for the respective *path* parameter.

56975 **RETURN VALUE**

56976 CX Upon successful completion, the *rename()* function shall return 0. Otherwise, it shall return -1,  
56977 *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by  
56978 *new* shall be changed or created.

56979 CX Upon successful completion, the *renameat()* function shall return 0. Otherwise, it shall return -1  
56980 and set *errno* to indicate the error.

56981 **ERRORS**

56982 CX The *rename()* and *renameat()* functions shall fail if:

56983 CX [EACCES] A component of either path prefix denies search permission; or one of the  
56984 directories containing *old* or *new* denies write permissions; or, write  
56985 permission is required and is denied for a directory pointed to by the *old* or  
56986 *new* arguments.

56987 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another  
56988 process, and the implementation considers this an error.

56989 CX [EEXIST] or [ENOTEMPTY]  
56990 The link named by *new* is a directory that is not an empty directory.

56991 CX [EINVAL] The *old* pathname names an ancestor directory of the *new* pathname, or either  
56992 *pathname* argument contains a final component that is dot or dot-dot.

56993 CX [EIO] A physical I/O error has occurred.

56994 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file  
56995 that is not a directory.

56996 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
56997 argument.

56998 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory  
56999 of *new* would exceed {LINK\_MAX}.

57000 CX [ENAMETOOLONG]  
57001 The length of a component of a pathname is longer than {NAME\_MAX}.

57002 CX [ENOENT] The link named by *old* does not name an existing file, a component of the path  
57003 prefix of *new* does not exist, or either *old* or *new* points to an empty string.

57004	CX	[ENOSPC]	The directory that would contain <i>new</i> cannot be extended.
57005	CX	[ENOTDIR]	A component of either path prefix is not a directory; or the <i>old</i> argument names a directory and the <i>new</i> argument names a non-directory file; or the <i>old</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>new</i> argument names a non-existent file, contains at least one non- <code>&lt;slash&gt;</code> character, and ends with one or more trailing <code>&lt;slash&gt;</code> characters.
57006			
57007			
57008			
57009			
57010			
57011			
57012	XSI	[EPERM] or [EACCES]	The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by <i>old</i> and the process does not satisfy the criteria specified in XBD Section 4.2 (on page 107) with respect to <i>old</i> ; or <i>new</i> refers to an existing file, the <code>S_ISVTX</code> flag is set on the directory containing this file, and the process does not satisfy the criteria specified in XBD Section 4.2 with respect to this file.
57013			
57014			
57015			
57016			
57017			
57018	CX	[EROFS]	The requested operation requires writing in a directory on a read-only file system.
57019			
57020	CX	[EXDEV]	The links named by <i>new</i> and <i>old</i> are on different file systems and the implementation does not support links between file systems.
57021			
57022	CX		In addition, the <code>renameat()</code> function shall fail if:
57023		[EACCES]	<i>oldfd</i> or <i>newfd</i> was not opened with <code>O_SEARCH</code> and the permissions of the directory underlying <i>oldfd</i> or <i>newfd</i> respectively do not permit directory searches.
57024			
57025			
57026		[EBADF]	The <i>old</i> argument does not specify an absolute path and the <i>oldfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching, or the <i>new</i> argument does not specify an absolute path and the <i>newfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
57027			
57028			
57029			
57030			
57031	CX		The <code>rename()</code> and <code>renameat()</code> functions may fail if:
57032	OB XSR	[EBUSY]	The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.
57033	CX	[ELOOP]	More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>path</i> argument.
57034			
57035	CX	[ENAMETOOLONG]	The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .
57036			
57037			
57038			
57039	CX	[ETXTBSY]	The file to be renamed is a pure procedure (shared text) file that is being executed.
57040			
57041	CX		The <code>renameat()</code> function may fail if:
57042		[ENOTDIR]	The <i>old</i> argument is not an absolute path and <i>oldfd</i> is neither <code>AT_FDCWD</code> nor a file descriptor associated with a directory, or the <i>new</i> argument is not an absolute path and <i>newfd</i> is neither <code>AT_FDCWD</code> nor a file descriptor associated with a directory.
57043			
57044			
57045			

**rename()**57046 **EXAMPLES**57047 **Renaming a File**

57048 The following example shows how to rename a file named `/home/cnd/mod1` to  
 57049 `/home/cnd/mod2`.

```
57050 #include <stdio.h>
57051 int status;
57052 ...
57053 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

57054 **APPLICATION USAGE**

57055 Some implementations mark for update the last file status change timestamp of renamed files  
 57056 and some do not. Applications which make use of the last file status change timestamp may  
 57057 behave differently with respect to renamed files unless they are designed to allow for either  
 57058 behavior.

57059 **RATIONALE**

57060 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its  
 57061 inclusion here expands that definition to include actions on directories and specifies behavior  
 57062 when the *new* parameter names a file that already exists. That specification requires that the  
 57063 action of the function be atomic.

57064 One of the reasons for introducing this function was to have a means of renaming directories  
 57065 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,  
 57066 thus constraining links to directories to those made by *mkdir()*.

57067 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
57068 rename("x", "x");
```

57069 does not remove the file.

57070 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

57071 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in  
 57072 *unlink()*. For a discussion of [EXDEV], see *link()*.

57073 The purpose of the *renameat()* function is to rename files in directories other than the current  
 57074 working directory without exposure to race conditions. Any part of the path of a file could be  
 57075 changed in parallel to a call to *rename()*, resulting in unspecified behavior. By opening file  
 57076 descriptors for the source and target directories and using the *renameat()* function it can be  
 57077 guaranteed that that renamed file is located correctly and the resulting file is in the desired  
 57078 directory.

57079 **FUTURE DIRECTIONS**

57080 None.

57081 **SEE ALSO**

57082 *link()*, *rmdir()*, *symlink()*, *unlink()*

57083 XBD Section 4.2 (on page 107), `<stdio.h>`

57084 **CHANGE HISTORY**

57085 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

57086 **Issue 5**

57087 The [EBUSY] error is added to the optional part of the ERRORS section.

57088 **Issue 6**

57089 Extensions beyond the ISO C standard are marked.

57090 The following new requirements on POSIX implementations derive from alignment with the  
57091 Single UNIX Specification:

- 57092 • The [EIO] mandatory error condition is added.
- 57093 • The [ELOOP] mandatory error condition is added.
- 57094 • A second [ENAMETOOLONG] is added as an optional error condition.
- 57095 • The [ETXTBSY] optional error condition is added.

57096 The following changes were made to align with the IEEE P1003.1a draft standard:

- 57097 • Details are added regarding the treatment of symbolic links.
- 57098 • The [ELOOP] optional error condition is added.

57099 The normative text is updated to avoid use of the term “must” for application requirements.

57100 **Issue 7**

57101 Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the  
57102 [ENOTDIR] error.

57103 Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final  
57104 component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

57105 Austin Group Interpretation 1003.1-2001 #143 is applied.

57106 Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error  
57107 condition also applies to the case in which a component of *new* does not exist.

57108 Austin Group Interpretations 1003.1-2001 #174 and #181 are applied.

57109 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API  
57110 Set Part 2.

57111 Changes are made related to support for finegrained timestamps.

57112 Changes are made to allow a directory to be opened for searching.

**rewind()**57113 **NAME**57114 `rewind` — reset the file position indicator in a stream57115 **SYNOPSIS**57116 `#include <stdio.h>`57117 `void rewind(FILE *stream);`57118 **DESCRIPTION**57119 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
57120 conflict between the requirements described here and the ISO C standard is unintentional. This  
57121 volume of POSIX.1-2008 defers to the ISO C standard.

57122 The call:

57123 `rewind(stream)`

57124 shall be equivalent to:

57125 `(void) fseek(stream, 0L, SEEK_SET)`57126 except that `rewind()` shall also clear the error indicator.57127 CX Since `rewind()` does not return a value, an application wishing to detect errors should clear `errno`,  
57128 then call `rewind()`, and if `errno` is non-zero, assume an error has occurred.57129 **RETURN VALUE**57130 The `rewind()` function shall not return a value.57131 **ERRORS**57132 CX Refer to `fseek()` with the exception of [EINVAL] which does not apply.57133 **EXAMPLES**

57134 None.

57135 **APPLICATION USAGE**

57136 None.

57137 **RATIONALE**

57138 None.

57139 **FUTURE DIRECTIONS**

57140 None.

57141 **SEE ALSO**57142 `fseek()`57143 XBD `<stdio.h>`57144 **CHANGE HISTORY**

57145 First released in Issue 1. Derived from Issue 1 of the SVID.

57146 **Issue 6**

57147 Extensions beyond the ISO C standard are marked.

57148 **NAME**

57149            rewinddir — reset the position of a directory stream to the beginning of a directory

57150 **SYNOPSIS**

57151            #include &lt;dirent.h&gt;

57152            void rewinddir(DIR \*dirp);

57153 **DESCRIPTION**

57154            The *rewinddir()* function shall reset the position of the directory stream to which *dirp* refers to the beginning of the directory. It shall also cause the directory stream to refer to the current state of the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a directory stream, the effect is undefined.

57158            After a call to the *fork()* function, either the parent or child (but not both) may continue processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes use these functions, the result is undefined.

57161 **RETURN VALUE**57162            The *rewinddir()* function shall not return a value.57163 **ERRORS**

57164            No errors are defined.

57165 **EXAMPLES**

57166            None.

57167 **APPLICATION USAGE**

57168            The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to examine the contents of the directory. This method is recommended for portability.

57170 **RATIONALE**

57171            None.

57172 **FUTURE DIRECTIONS**

57173            None.

57174 **SEE ALSO**57175            *closedir()*, *fdopendir()*, *readdir()*

57176            XBD &lt;dirent.h&gt;, &lt;sys/types.h&gt;

57177 **CHANGE HISTORY**

57178            First released in Issue 2.

57179 **Issue 6**

57180            In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

57181            The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 57183            • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

**rint()**57186 **NAME**

57187 rint, rintf, rintl — round-to-nearest integral value

57188 **SYNOPSIS**

```
57189 #include <math.h>
57190 double rint(double x);
57191 float rintf(float x);
57192 long double rintl(long double x);
```

57193 **DESCRIPTION**

57194 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 57195 conflict between the requirements described here and the ISO C standard is unintentional. This  
 57196 volume of POSIX.1-2008 defers to the ISO C standard.

57197 These functions shall return the integral value (represented as a **double**) nearest  $x$  in the  
 57198 direction of the current rounding mode. The current rounding mode is implementation-defined.

57199 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to  
 57200 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be  
 57201 equivalent to *ceil()*.

57202 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that  
 57203 they may raise the inexact floating-point exception if the result differs in value from the  
 57204 argument.

57205 An application wishing to check for error situations should set *errno* to zero and call  
 57206 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 57207 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 57208 zero, an error has occurred.

57209 **RETURN VALUE**

57210 Upon successful completion, these functions shall return the integer (represented as a double  
 57211 precision number) nearest  $x$  in the direction of the current rounding mode.

57212 MX If  $x$  is NaN, a NaN shall be returned.

57213 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

57214 XSI If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*  
 57215 shall return the value of the macro  $\pm \text{HUGE\_VAL}$ ,  $\pm \text{HUGE\_VALF}$ , and  $\pm \text{HUGE\_VALL}$  (with the  
 57216 same sign as  $x$ ), respectively.

57217 **ERRORS**

57218 These functions shall fail if:

57219 XSI **Range Error** The result would cause an overflow.

57220 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 57221 then *errno* shall be set to [ERANGE]. If the integer expression  
 57222 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 57223 floating-point exception shall be raised.

57224 **EXAMPLES**

57225 None.

57226 **APPLICATION USAGE**57227 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
57228 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.57229 **RATIONALE**

57230 None.

57231 **FUTURE DIRECTIONS**

57232 None.

57233 **SEE ALSO**57234 *abs()*, *ceil()*, *feclearexcept()*, *fetetestexcept()*, *floor()*, *isnan()*, *nearbyint()*

57235 XBD Section 4.19 (on page 116), &lt;math.h&gt;

57236 **CHANGE HISTORY**

57237 First released in Issue 4, Version 2.

57238 **Issue 5**

57239 Moved from X/OPEN UNIX extension to BASE.

57240 **Issue 6**

57241 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 57242 • The *rintf()* and *rintl()* functions are added.
- 57243 • The *rint()* function is no longer marked as an extension.
- 57244 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
57245 revised to align with the ISO/IEC 9899:1999 standard.
- 57246 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard  
57247 are marked.

**rmdir()**57248 **NAME**57249 `rmdir` — remove a directory57250 **SYNOPSIS**

```
57251 #include <unistd.h>
57252 int rmdir(const char *path);
```

57253 **DESCRIPTION**

57254 The `rmdir()` function shall remove a directory whose name is given by *path*. The directory shall  
 57255 be removed only if it is an empty directory.

57256 If the directory is the root directory or the current working directory of any process, it is  
 57257 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

57258 If *path* names a symbolic link, then `rmdir()` shall fail and set *errno* to [ENOTDIR].

57259 If the *path* argument refers to a path whose final component is either dot or dot-dot, `rmdir()` shall  
 57260 fail.

57261 If the directory's link count becomes 0 and no process has the directory open, the space occupied  
 57262 by the directory shall be freed and the directory shall no longer be accessible. If one or more  
 57263 processes have the directory open when the last link is removed, the dot and dot-dot entries, if  
 57264 present, shall be removed before `rmdir()` returns and no new entries may be created in the  
 57265 directory, but the directory shall not be removed until all references to the directory are closed.

57266 If the directory is not an empty directory, `rmdir()` shall fail and set *errno* to [EEXIST] or  
 57267 [ENOTEMPTY].

57268 Upon successful completion, `rmdir()` shall mark for update the last data modification and last  
 57269 file status change timestamps of the parent directory.

57270 **RETURN VALUE**

57271 Upon successful completion, the function `rmdir()` shall return 0. Otherwise, -1 shall be returned,  
 57272 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.

57273 **ERRORS**

57274 The `rmdir()` function shall fail if:

57275 [EACCES] Search permission is denied on a component of the path prefix, or write  
 57276 permission is denied on the parent directory of the directory to be removed.

57277 [EBUSY] The directory to be removed is currently in use by the system or some process  
 57278 and the implementation considers this to be an error.

57279 [EEXIST] or [ENOTEMPTY]

57280 The *path* argument names a directory that is not an empty directory, or there  
 57281 are hard links to the directory other than dot or a single entry in dot-dot.

57282 [EINVAL] The *path* argument contains a last component that is dot.

57283 [EIO] A physical I/O error has occurred.

57284 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 57285 argument.

57286 [ENAMETOOLONG]

57287 The length of a component of a pathname is longer than {NAME\_MAX}.

57288 [ENOENT] A component of *path* does not name an existing file, or the *path* argument  
 57289 names a nonexistent directory or points to an empty string.

- 57290 [ENOTDIR] A component of *path* is not a directory.
- 57291 XSI [EPERM] or [EACCES]  
 57292 The S\_ISVTX flag is set on the directory containing the file referred to by the  
 57293 *path* argument and the process does not satisfy the criteria specified in XBD  
 57294 Section 4.2 (on page 107).
- 57295 [EROFS] The directory entry to be removed resides on a read-only file system.
- 57296 The *rmdir()* function may fail if:
- 57297 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 57298 resolution of the *path* argument.
- 57299 [ENAMETOOLONG]  
 57300 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 57301 symbolic link produced an intermediate result with a length that exceeds  
 57302 {PATH\_MAX}.

57303 **EXAMPLES**57304 **Removing a Directory**

57305 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
57306 #include <unistd.h>
57307 int status;
57308 ...
57309 status = rmdir("/home/cnd/mod1");
```

57310 **APPLICATION USAGE**

57311 None.

57312 **RATIONALE**

57313 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the  
 57314 condition when the directory to be removed does not exist or *new* already exists. When the 1984  
 57315 /usr/group standard was published, it contained [EEXIST] instead. When these functions were  
 57316 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore,  
 57317 several existing applications and implementations support/use both forms, and no agreement  
 57318 could be reached on either value. All implementations are required to supply both [EEXIST] and  
 57319 [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-  
 57320 language `case` statements.

57321 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the  
 57322 parent directory to be removed is not clear, particularly in the presence of multiple links to a  
 57323 directory.

57324 The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are  
 57325 multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or  
 57326 [ENOTEMPTY] clarifies the behavior in this case.

57327 If the current working directory of the process is being removed, that should be an allowed  
 57328 error.

57329 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in  
 57330 Section 2.3 (on page 477) about returning any one of the possible errors permits that behavior to  
 57331 continue. The [ELOOP] error may be returned if more than {SYMLOOP\_MAX} symbolic links  
 57332 are encountered during resolution of the *path* argument.

**rmdir()**57333 **FUTURE DIRECTIONS**

57334 None.

57335 **SEE ALSO**57336 [Section 2.3](#) (on page 477), *mkdir()*, *remove()*, *rename()*, *unlink()*57337 XBD [Section 4.2](#) (on page 107), `<unistd.h>`57338 **CHANGE HISTORY**

57339 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

57340 **Issue 6**57341 The following new requirements on POSIX implementations derive from alignment with the  
57342 Single UNIX Specification:

- 57343 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 57344 • The [EIO] mandatory error condition is added.
- 57345 • The [ELOOP] mandatory error condition is added.
- 57346 • A second [ENAMETOOLONG] is added as an optional error condition.

57347 The following changes were made to align with the IEEE P1003.1a draft standard:

- 57348 • The [ELOOP] optional error condition is added.

57349 **Issue 7**

57350 Austin Group Interpretation 1003.1-2001 #143 is applied.

57351 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for  
57352 operations when the S\_ISVTX bit is set.

57353 Changes are made related to support for finegrained timestamps.

57354 **NAME**

57355 round, roundf, roundl — round to the nearest integer value in a floating-point format

57356 **SYNOPSIS**

```
57357 #include <math.h>
57358 double round(double x);
57359 float roundf(float x);
57360 long double roundl(long double x);
```

57361 **DESCRIPTION**

57362 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 57363 conflict between the requirements described here and the ISO C standard is unintentional. This  
 57364 volume of POSIX.1-2008 defers to the ISO C standard.

57365 These functions shall round their argument to the nearest integer value in floating-point format,  
 57366 rounding halfway cases away from zero, regardless of the current rounding direction.

57367 An application wishing to check for error situations should set *errno* to zero and call  
 57368 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 57369 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 57370 zero, an error has occurred.

57371 **RETURN VALUE**

57372 Upon successful completion, these functions shall return the rounded integer value.

57373 MX If *x* is NaN, a NaN shall be returned.57374 If *x* is  $\pm 0$  or  $\pm \text{Inf}$ , *x* shall be returned.

57375 XSI If the correct value would cause overflow, a range error shall occur and *round*(*x*), *roundf*(*x*), and  
 57376 *roundl*(*x*) shall return the value of the macro  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$   
 57377 (with the same sign as *x*), respectively.

57378 **ERRORS**

57379 These functions may fail if:

57380 XSI **Range Error** The result overflows.

57381 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 57382 then *errno* shall be set to [ERANGE]. If the integer expression  
 57383 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 57384 floating-point exception shall be raised.

57385 **EXAMPLES**

57386 None.

57387 **APPLICATION USAGE**

57388 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 57389 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

57390 **RATIONALE**

57391 None.

57392 **FUTURE DIRECTIONS**

57393 None.

**round()***System Interfaces*57394 **SEE ALSO**57395 *feclearexcept()*, *fetestexcept()*57396 XBD Section 4.19 (on page 116), [<math.h>](#)57397 **CHANGE HISTORY**

57398 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57399 **NAME**

57400 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT\_RADIX

57401 **SYNOPSIS**

```
57402 #include <math.h>
57403 double scalbn(double x, long n);
57404 float scalblnf(float x, long n);
57405 long double scalblnl(long double x, long n);
57406 double scalbn(double x, int n);
57407 float scalbnf(float x, int n);
57408 long double scalbnl(long double x, int n);
```

57409 **DESCRIPTION**

57410 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 57411 conflict between the requirements described here and the ISO C standard is unintentional. This  
 57412 volume of POSIX.1-2008 defers to the ISO C standard.

57413 These functions shall compute  $x * FLT\_RADIX^n$  efficiently, not normally by computing  
 57414  $FLT\_RADIX^n$  explicitly.

57415 An application wishing to check for error situations should set *errno* to zero and call  
 57416 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 57417 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 57418 zero, an error has occurred.

57419 **RETURN VALUE**

57420 Upon successful completion, these functions shall return  $x * FLT\_RADIX^n$ .

57421 If the result would cause overflow, a range error shall occur and these functions shall return  
 57422  $\pm HUGE\_VAL$ ,  $\pm HUGE\_VALF$ , and  $\pm HUGE\_VALL$  (according to the sign of *x*) as appropriate for  
 57423 the return type of the function.

57424 If the correct value would cause underflow, and is not representable, a range error may occur,  
 57425 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

57426 **MX** If *x* is NaN, a NaN shall be returned.

57427 If *x* is  $\pm 0$  or  $\pm Inf$ , *x* shall be returned.

57428 If *n* is 0, *x* shall be returned.

57429 If the correct value would cause underflow, and is representable, a range error may occur and  
 57430 the correct value shall be returned.

57431 **ERRORS**

57432 These functions shall fail if:

57433 **Range Error** The result overflows.

57434 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 57435 then *errno* shall be set to [ERANGE]. If the integer expression  
 57436 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 57437 floating-point exception shall be raised.

57438 These functions may fail if:

57439 **Range Error** The result underflows.

57440 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 57441 then *errno* shall be set to [ERANGE]. If the integer expression

**scalbln()**

57442 *(math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
57443 floating-point exception shall be raised.

**EXAMPLES**

57444 None.

**APPLICATION USAGE**

57447 On error, the expressions *(math\_errhandling* & MATH\_ERRNO) and *(math\_errhandling* &  
57448 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**RATIONALE**

57450 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*  
57451 function from the Single UNIX Specification. The difference is that the *scalb()* function has a  
57452 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.  
57453 The three functions whose second type is **long** are provided because the factor required to scale  
57454 from the smallest positive floating-point value to the largest finite one, on many  
57455 implementations, is too large to represent in the minimum-width **int** format.

**FUTURE DIRECTIONS**

57456 None.

**SEE ALSO**

57459 *feclearexcept()*, *fetestexcept()*

57460 XBD Section 4.19 (on page 116), [<math.h>](#)

**CHANGE HISTORY**

57461 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.  
57462

57463 **NAME**

57464       scandir — scan a directory

57465 **SYNOPSIS**

57466       #include &lt;dirent.h&gt;

```
57467       int scandir(const char *dir, struct dirent ***namelist,  
57468                   int (*sel)(const struct dirent *),  
57469                   int (*compar)(const struct dirent **, const struct dirent **));
```

57470 **DESCRIPTION**57471       Refer to *alphasort()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**scanf()**57472 **NAME**

57473 scanf — convert formatted input

57474 **SYNOPSIS**

57475 #include &lt;stdio.h&gt;

57476 int scanf(const char \*restrict *format*, ...);57477 **DESCRIPTION**57478 Refer to *fscanf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57479 **NAME**

57480 sched\_get\_priority\_max, sched\_get\_priority\_min — get priority limits (REALTIME)

57481 **SYNOPSIS**

```
57482 PS|TPS #include <sched.h>
57483 int sched_get_priority_max(int policy);
57484 int sched_get_priority_min(int policy);
```

57485 **DESCRIPTION**57486 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the appropriate  
57487 maximum or minimum, respectively, for the scheduling policy specified by *policy*.57488 The value of *policy* shall be one of the scheduling policy values defined in `<sched.h>`.57489 **RETURN VALUE**57490 If successful, the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the  
57491 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a  
57492 value of `-1` and set *errno* to indicate the error.57493 **ERRORS**57494 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall fail if:57495 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling  
57496 policy.57497 **EXAMPLES**

57498 None.

57499 **APPLICATION USAGE**

57500 None.

57501 **RATIONALE**

57502 None.

57503 **FUTURE DIRECTIONS**

57504 None.

57505 **SEE ALSO**57506 *sched\_getparam()*, *sched\_setparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
57507 *sched\_setscheduler()*.57508 XBD `<sched.h>`57509 **CHANGE HISTORY**

57510 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57511 **Issue 6**

57512 These functions are marked as part of the Process Scheduling option.

57513 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
57514 implementation does not support the Process Scheduling option.57515 The [ESRCH] error condition has been removed since these functions do not take a *pid*  
57516 argument.57517 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin  
57518 code in the SYNOPSIS to PS|TPS.

**sched\_getparam()**57519 **NAME**57520 sched\_getparam — get scheduling parameters (**REALTIME**)57521 **SYNOPSIS**

```
57522 PS #include <sched.h>
57523 int sched_getparam(pid_t pid, struct sched_param *param);
```

57524 **DESCRIPTION**

57525 The *sched\_getparam()* function shall return the scheduling parameters of a process specified by  
57526 *pid* in the **sched\_param** structure pointed to by *param*.

57527 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
57528 parameters for the process whose process ID is equal to *pid* shall be returned.

57529 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of  
57530 the *sched\_getparam()* function is unspecified if the value of *pid* is negative.

57531 **RETURN VALUE**

57532 Upon successful completion, the *sched\_getparam()* function shall return zero. If the call to  
57533 *sched\_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate  
57534 the error.

57535 **ERRORS**

57536 The *sched\_getparam()* function shall fail if:

57537 [EPERM] The requesting process does not have permission to obtain the scheduling  
57538 parameters of the specified process.

57539 [ESRCH] No process can be found corresponding to that specified by *pid*.

57540 **EXAMPLES**

57541 None.

57542 **APPLICATION USAGE**

57543 None.

57544 **RATIONALE**

57545 None.

57546 **FUTURE DIRECTIONS**

57547 None.

57548 **SEE ALSO**

57549 *sched\_getscheduler()*, *sched\_setparam()*, *sched\_setscheduler()*

57550 XBD <**sched.h**>

57551 **CHANGE HISTORY**

57552 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57553 **Issue 6**

57554 The *sched\_getparam()* function is marked as part of the Process Scheduling option.

57555 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
57556 implementation does not support the Process Scheduling option.

57557 **NAME**57558 sched\_getscheduler — get scheduling policy (**REALTIME**)57559 **SYNOPSIS**

```
57560 PS #include <sched.h>
57561 int sched_getscheduler(pid_t pid);
```

57562 **DESCRIPTION**

57563 The *sched\_getscheduler()* function shall return the scheduling policy of the process specified by  
 57564 *pid*. If the value of *pid* is negative, the behavior of the *sched\_getscheduler()* function is  
 57565 unspecified.

57566 The values that can be returned by *sched\_getscheduler()* are defined in the `<sched.h>` header.

57567 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 57568 policy shall be returned for the process whose process ID is equal to *pid*.

57569 If *pid* is zero, the scheduling policy shall be returned for the calling process.

57570 **RETURN VALUE**

57571 Upon successful completion, the *sched\_getscheduler()* function shall return the scheduling policy  
 57572 of the specified process. If unsuccessful, the function shall return `-1` and set *errno* to indicate the  
 57573 error.

57574 **ERRORS**

57575 The *sched\_getscheduler()* function shall fail if:

- |       |         |  |
|-------|---------|--|
| 57576 | [EPERM] | The requesting process does not have permission to determine the scheduling policy of the specified process. |
| 57577 |         |  |
| 57578 | [ESRCH] | No process can be found corresponding to that specified by <i>pid</i> .                                      |

57579 **EXAMPLES**

57580 None.

57581 **APPLICATION USAGE**

57582 None.

57583 **RATIONALE**

57584 None.

57585 **FUTURE DIRECTIONS**

57586 None.

57587 **SEE ALSO**

57588 [sched\\_getparam\(\)](#), [sched\\_setparam\(\)](#), [sched\\_setscheduler\(\)](#)

57589 XBD `<sched.h>`

57590 **CHANGE HISTORY**

57591 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57592 **Issue 6**

57593 The *sched\_getscheduler()* function is marked as part of the Process Scheduling option.

57594 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 57595 implementation does not support the Process Scheduling option.

**sched\_rr\_get\_interval()**

System Interfaces

57596 **NAME**57597 sched\_rr\_get\_interval — get execution time limits (**REALTIME**)57598 **SYNOPSIS**

```
57599 PS|TPS #include <sched.h>
57600 int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

57601 **DESCRIPTION**

57602 The *sched\_rr\_get\_interval()* function shall update the **timespec** structure referenced by the  
 57603 *interval* argument to contain the current execution time limit (that is, time quantum) for the  
 57604 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process  
 57605 shall be returned.

57606 **RETURN VALUE**

57607 If successful, the *sched\_rr\_get\_interval()* function shall return zero. Otherwise, it shall return a  
 57608 value of  $-1$  and set *errno* to indicate the error.

57609 **ERRORS**

57610 The *sched\_rr\_get\_interval()* function shall fail if:

57611 [ESRCH] No process can be found corresponding to that specified by *pid*.

57612 **EXAMPLES**

57613 None.

57614 **APPLICATION USAGE**

57615 None.

57616 **RATIONALE**

57617 None.

57618 **FUTURE DIRECTIONS**

57619 None.

57620 **SEE ALSO**

57621 [sched\\_getparam\(\)](#), [sched\\_get\\_priority\\_max\(\)](#), [sched\\_getscheduler\(\)](#), [sched\\_setparam\(\)](#),  
 57622 [sched\\_setscheduler\(\)](#)

57623 XBD [<sched.h>](#)

57624 **CHANGE HISTORY**

57625 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57626 **Issue 6**

57627 The *sched\_rr\_get\_interval()* function is marked as part of the Process Scheduling option.

57628 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 57629 implementation does not support the Process Scheduling option.

57630 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in  
 57631 the SYNOPSIS to PS|TPS.

57632 **NAME**57633 sched\_setparam — set scheduling parameters (**REALTIME**)57634 **SYNOPSIS**

```
57635 PS #include <sched.h>
57636 int sched_setparam(pid_t pid, const struct sched_param *param);
```

57637 **DESCRIPTION**

57638 The *sched\_setparam()* function shall set the scheduling parameters of the process specified by *pid*  
 57639 to the values specified by the **sched\_param** structure pointed to by *param*. The value of the  
 57640 *sched\_priority* member in the **sched\_param** structure shall be any integer within the inclusive  
 57641 priority range for the current scheduling policy of the process specified by *pid*. Higher  
 57642 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the  
 57643 behavior of the *sched\_setparam()* function is unspecified.

57644 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 57645 parameters shall be set for the process whose process ID is equal to *pid*.

57646 If *pid* is zero, the scheduling parameters shall be set for the calling process.

57647 The conditions under which one process has permission to change the scheduling parameters of  
 57648 another process are implementation-defined.

57649 Implementations may require the requesting process to have appropriate privileges to set its  
 57650 own scheduling parameters or those of another process.

57651 See [Scheduling Policies](#) (on page 501) for a description on how this function affects the  
 57652 scheduling of the threads within the target process.

57653 SS If the current scheduling policy for the target process is not SCHED\_FIFO, SCHED\_RR, or  
 57654 SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
 57655 SCHED\_OTHER policy.

57656 SS The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 57657 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

57658 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1, {SS\_REPL\_MAX}] for the  
 57659 function to succeed; if not, the function shall fail. It is unspecified whether the  
 57660 *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are stored as provided by this function or are  
 57661 rounded to align with the resolution of the clock being used.

57662 This function is not atomic with respect to other threads in the process. Threads may continue to  
 57663 execute while this function call is in the process of changing the scheduling policy for the  
 57664 underlying kernel-scheduled entities used by the process contention scope threads.

57665 **RETURN VALUE**

57666 If successful, the *sched\_setparam()* function shall return zero.

57667 If the call to *sched\_setparam()* is unsuccessful, the priority shall remain unchanged, and the  
 57668 function shall return a value of  $-1$  and set *errno* to indicate the error.

57669 **ERRORS**

57670 The *sched\_setparam()* function shall fail if:

57671 [EINVAL] One or more of the requested scheduling parameters is outside the range  
 57672 defined for the scheduling policy of the specified *pid*.

**sched\_setparam()**

System Interfaces

57673	[EPERM]	The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have appropriate privileges to invoke <i>sched_setparam()</i> .
57674		
57675		
57676	[ESRCH]	No process can be found corresponding to that specified by <i>pid</i> .
57677	<b>EXAMPLES</b>	
57678		None.
57679	<b>APPLICATION USAGE</b>	
57680		None.
57681	<b>RATIONALE</b>	
57682		None.
57683	<b>FUTURE DIRECTIONS</b>	
57684		None.
57685	<b>SEE ALSO</b>	
57686		<a href="#">Scheduling Policies</a> (on page 501), <i>sched_getparam()</i> , <i>sched_getscheduler()</i> , <i>sched_setscheduler()</i>
57687		XBD <a href="#">&lt;sched.h&gt;</a>
57688	<b>CHANGE HISTORY</b>	
57689		First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
57690	<b>Issue 6</b>	
57691		The <i>sched_setparam()</i> function is marked as part of the Process Scheduling option.
57692		The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.
57693		
57694		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
57695		
57696		<ul style="list-style-type: none"> <li>• In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is added.</li> </ul>
57697		
57698		<ul style="list-style-type: none"> <li>• Sections describing two-level scheduling and atomicity of the function are added.</li> </ul>
57699		The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
57700		IEEE PASC Interpretation 1003.1 #100 is applied.
57701	<b>Issue 7</b>	
57702		Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.
57703		Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the <i>sched_ss_repl_period</i> and <i>sched_ss_init_budget</i> values.
57704		

57705 **NAME**57706 sched\_setscheduler — set scheduling policy and parameters (**REALTIME**)57707 **SYNOPSIS**

```
57708 PS #include <sched.h>
57709
57709 int sched_setscheduler(pid_t pid, int policy,
57710 const struct sched_param *param);
```

57711 **DESCRIPTION**

57712 The `sched_setscheduler()` function shall set the scheduling policy and scheduling parameters of  
 57713 the process specified by `pid` to `policy` and the parameters specified in the `sched_param` structure  
 57714 pointed to by `param`, respectively. The value of the `sched_priority` member in the `sched_param`  
 57715 structure shall be any integer within the inclusive priority range for the scheduling policy  
 57716 specified by `policy`. If the value of `pid` is negative, the behavior of the `sched_setscheduler()`  
 57717 function is unspecified.

57718 The possible values for the `policy` parameter are defined in the `<sched.h>` header.

57719 If a process specified by `pid` exists, and if the calling process has permission, the scheduling  
 57720 policy and scheduling parameters shall be set for the process whose process ID is equal to `pid`.

57721 If `pid` is zero, the scheduling policy and scheduling parameters shall be set for the calling  
 57722 process.

57723 The conditions under which one process has appropriate privileges to change the scheduling  
 57724 parameters of another process are implementation-defined.

57725 Implementations may require that the requesting process have permission to set its own  
 57726 scheduling parameters or those of another process. Additionally, implementation-defined  
 57727 restrictions may apply as to the appropriate privileges required to set the scheduling policy of  
 57728 the process, or the scheduling policy of another process, to a particular value.

57729 The `sched_setscheduler()` function shall be considered successful if it succeeds in setting the  
 57730 scheduling policy and scheduling parameters of the process specified by `pid` to the values  
 57731 specified by `policy` and the structure pointed to by `param`, respectively.

57732 See [Scheduling Policies](#) (on page 501) for a description on how this function affects the  
 57733 scheduling of the threads within the target process.

57734 SS If the current scheduling policy for the target process is not `SCHED_FIFO`, `SCHED_RR`, or  
 57735 `SCHED_SPORADIC`, the result is implementation-defined; this case includes the  
 57736 `SCHED_OTHER` policy.

57737 SS The specified `sched_ss_repl_period` shall be greater than or equal to the specified  
 57738 `sched_ss_init_budget` for the function to succeed; if it is not, then the function shall fail.

57739 The value of `sched_ss_max_repl` shall be within the inclusive range `[1, {SS_REPL_MAX}]` for the  
 57740 function to succeed; if not, the function shall fail. It is unspecified whether the  
 57741 `sched_ss_repl_period` and `sched_ss_init_budget` values are stored as provided by this function or are  
 57742 rounded to align with the resolution of the clock being used.

57743 This function is not atomic with respect to other threads in the process. Threads may continue to  
 57744 execute while this function call is in the process of changing the scheduling policy and  
 57745 associated scheduling parameters for the underlying kernel-scheduled entities used by the  
 57746 process contention scope threads.

**sched\_setscheduler()**

System Interfaces

57747 **RETURN VALUE**

57748 Upon successful completion, the function shall return the former scheduling policy of the  
 57749 specified process. If the *sched\_setscheduler()* function fails to complete successfully, the policy  
 57750 and scheduling parameters shall remain unchanged, and the function shall return a value of -1  
 57751 and set *errno* to indicate the error.

57752 **ERRORS**

57753 The *sched\_setscheduler()* function shall fail if:

57754 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters  
 57755 contained in *param* is outside the valid range for the specified scheduling  
 57756 policy.

57757 [EPERM] The requesting process does not have permission to set either or both of the  
 57758 scheduling parameters or the scheduling policy of the specified process.

57759 [ESRCH] No process can be found corresponding to that specified by *pid*.

57760 **EXAMPLES**

57761 None.

57762 **APPLICATION USAGE**

57763 None.

57764 **RATIONALE**

57765 None.

57766 **FUTURE DIRECTIONS**

57767 None.

57768 **SEE ALSO**

57769 [Scheduling Policies](#) (on page 501), [sched\\_getparam\(\)](#), [sched\\_getscheduler\(\)](#), [sched\\_setparam\(\)](#)

57770 XBD [<sched.h>](#)

57771 **CHANGE HISTORY**

57772 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57773 **Issue 6**

57774 The *sched\_setscheduler()* function is marked as part of the Process Scheduling option.

57775 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 57776 implementation does not support the Process Scheduling option.

57777 The following new requirements on POSIX implementations derive from alignment with the  
 57778 Single UNIX Specification:

- 57779 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
 57780 added.

- 57781 • Sections describing two-level scheduling and atomicity of the function are added.

57782 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

57783 **Issue 7**

57784 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

57785 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
 57786 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.

57787 **NAME**

57788 sched\_yield — yield the processor

57789 **SYNOPSIS**

57790 #include &lt;sched.h&gt;

57791 int sched\_yield(void);

57792 **DESCRIPTION**57793 The *sched\_yield()* function shall force the running thread to relinquish the processor until it again  
57794 becomes the head of its thread list. It takes no arguments.57795 **RETURN VALUE**57796 The *sched\_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a  
57797 value of -1 and set *errno* to indicate the error.57798 **ERRORS**

57799 No errors are defined.

57800 **EXAMPLES**

57801 None.

57802 **APPLICATION USAGE**57803 The conceptual model for scheduling semantics in POSIX.1-2008 defines a set of thread lists. This  
57804 set of thread lists is always present regardless of the scheduling options supported by the  
57805 system. On a system where the Process Scheduling option is not supported, portable  
57806 applications should not make any assumptions regarding whether threads from other processes  
57807 will be on the same thread list.57808 **RATIONALE**

57809 None.

57810 **FUTURE DIRECTIONS**

57811 None.

57812 **SEE ALSO**

57813 XBD &lt;sched.h&gt;

57814 **CHANGE HISTORY**57815 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
57816 POSIX Threads Extension.57817 **Issue 6**57818 The *sched\_yield()* function is now marked as part of the Process Scheduling and Threads options.57819 **Issue 7**

57820 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

57821 The *sched\_yield()* function is moved to the Base.

**seed48()***System Interfaces*57822 **NAME**

57823 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

57824 **SYNOPSIS**

```
57825 XSI #include <stdlib.h>  
57826 unsigned short *seed48(unsigned short seed16v[3]);
```

57827 **DESCRIPTION**57828 Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57829 **NAME**

57830 seekdir — set the position of a directory stream

57831 **SYNOPSIS**

```
57832 XSI #include <dirent.h>
57833 void seekdir(DIR *dirp, long loc);
```

57834 **DESCRIPTION**

57835 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory  
 57836 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been  
 57837 returned from an earlier call to *telldir()* using the same directory stream. The new position  
 57838 reverts to the one associated with the directory stream when *telldir()* was performed.

57839 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*  
 57840 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to  
 57841 *readdir()* are unspecified.

57842 **RETURN VALUE**57843 The *seekdir()* function shall not return a value.57844 **ERRORS**

57845 No errors are defined.

57846 **EXAMPLES**

57847 None.

57848 **APPLICATION USAGE**

57849 None.

57850 **RATIONALE**

57851 The original standard developers perceived that there were restrictions on the use of the  
 57852 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these  
 57853 functions need not be supported on all POSIX-conforming systems. They are required on  
 57854 implementations supporting the XSI option.

57855 One of the perceived problems of implementation is that returning to a given point in a directory  
 57856 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-  
 57857 trees, hashing functions, or other similar mechanisms to order their directories are considered.  
 57858 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a  
 57859 given directory entry will be seen at all, or more than once.

57860 On systems not supporting these functions, their capability can sometimes be accomplished by  
 57861 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to  
 57862 relocate the position from which the filename was saved.

57863 **FUTURE DIRECTIONS**

57864 None.

57865 **SEE ALSO**57866 *fdopendir()*, *readdir()*, *telldir()*57867 XBD *<dirent.h>*, *<sys/types.h>*57868 **CHANGE HISTORY**

57869 First released in Issue 2.

**seekdir()**57870 **Issue 6**

57871 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

57872 **Issue 7**

57873 SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should  
57874 have been returned from an earlier call to *telldir()* using the same directory stream.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***select()**57875 **NAME**57876 `select` — synchronous I/O multiplexing57877 **SYNOPSIS**57878 `#include <sys/select.h>`

```
57879 int select(int nfds, fd_set *restrict readfds,  
57880           fd_set *restrict writefds, fd_set *restrict errorfds,  
57881           struct timeval *restrict timeout);
```

57882 **DESCRIPTION**57883 Refer to *pselect()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_close()**57884 **NAME**57885 `sem_close` — close a named semaphore57886 **SYNOPSIS**57887 `#include <semaphore.h>`  
57888 `int sem_close(sem_t *sem);`57889 **DESCRIPTION**57890 The `sem_close()` function shall indicate that the calling process is finished using the named  
57891 semaphore indicated by `sem`. The effects of calling `sem_close()` for an unnamed semaphore (one  
57892 created by `sem_init()`) are undefined. The `sem_close()` function shall deallocate (that is, make  
57893 available for reuse by a subsequent `sem_open()` by this process) any system resources allocated  
57894 by the system for use by this process for this semaphore. The effect of subsequent use of the  
57895 semaphore indicated by `sem` by this process is undefined. If the semaphore has not been  
57896 removed with a successful call to `sem_unlink()`, then `sem_close()` has no effect on the state of the  
57897 semaphore. If the `sem_unlink()` function has been successfully invoked for `name` after the most  
57898 recent call to `sem_open()` with `O_CREAT` for this semaphore, then when all processes that have  
57899 opened the semaphore close it, the semaphore is no longer accessible.57900 **RETURN VALUE**57901 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be  
57902 returned and `errno` set to indicate the error.57903 **ERRORS**57904 The `sem_close()` function may fail if:57905 `[EINVAL]` The `sem` argument is not a valid semaphore descriptor.57906 **EXAMPLES**

57907 None.

57908 **APPLICATION USAGE**57909 The `sem_close()` function is part of the Semaphores option and need not be available on all  
57910 implementations.57911 **RATIONALE**

57912 None.

57913 **FUTURE DIRECTIONS**

57914 None.

57915 **SEE ALSO**57916 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`, `sem_unlink()`57917 XBD `<semaphore.h>`57918 **CHANGE HISTORY**

57919 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57920 **Issue 6**57921 The `sem_close()` function is marked as part of the Semaphores option.57922 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an  
57923 implementation does not support the Semaphores option.57924 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS  
57925 section so that the `[EINVAL]` error becomes optional.

57926 **Issue 7**  
57927

The *sem\_close()* function is moved from the Semaphores option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_destroy()**57928 **NAME**57929 `sem_destroy` — destroy an unnamed semaphore57930 **SYNOPSIS**57931 `#include <semaphore.h>`  
57932 `int sem_destroy(sem_t *sem);`57933 **DESCRIPTION**57934 The `sem_destroy()` function shall destroy the unnamed semaphore indicated by `sem`. Only a  
57935 semaphore that was created using `sem_init()` may be destroyed using `sem_destroy()`; the effect of  
57936 calling `sem_destroy()` with a named semaphore is undefined. The effect of subsequent use of the  
57937 semaphore `sem` is undefined until `sem` is reinitialized by another call to `sem_init()`.57938 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The  
57939 effect of destroying a semaphore upon which other threads are currently blocked is undefined.57940 **RETURN VALUE**57941 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be  
57942 returned and `errno` set to indicate the error.57943 **ERRORS**57944 The `sem_destroy()` function may fail if:

- 57945 [EINVAL] The
- `sem`
- argument is not a valid semaphore.
- 
- 57946 [EBUSY] There are currently processes blocked on the semaphore.

57947 **EXAMPLES**

57948 None.

57949 **APPLICATION USAGE**57950 The `sem_destroy()` function is part of the Semaphores option and need not be available on all  
57951 implementations.57952 **RATIONALE**

57953 None.

57954 **FUTURE DIRECTIONS**

57955 None.

57956 **SEE ALSO**57957 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`57958 XBD `<semaphore.h>`57959 **CHANGE HISTORY**

57960 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57961 **Issue 6**57962 The `sem_destroy()` function is marked as part of the Semaphores option.57963 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
57964 implementation does not support the Semaphores option.57965 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS  
57966 section so that the [EINVAL] error becomes optional.

57967 **Issue 7**  
57968

The *sem\_destroy()* function is moved from the Semaphores option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_getvalue()**57969 **NAME**57970 `sem_getvalue` — get the value of a semaphore57971 **SYNOPSIS**57972 `#include <semaphore.h>`  
57973 `int sem_getvalue(sem_t *restrict sem, int *restrict sval);`57974 **DESCRIPTION**57975 The `sem_getvalue()` function shall update the location referenced by the `sval` argument to have  
57976 the value of the semaphore referenced by `sem` without affecting the state of the semaphore. The  
57977 updated value represents an actual semaphore value that occurred at some unspecified time  
57978 during the call, but it need not be the actual value of the semaphore when it is returned to the  
57979 calling process.57980 If `sem` is locked, then the object to which `sval` points shall either be set to zero or to a negative  
57981 number whose absolute value represents the number of processes waiting for the semaphore at  
57982 some unspecified time during the call.57983 **RETURN VALUE**57984 Upon successful completion, the `sem_getvalue()` function shall return a value of zero. Otherwise,  
57985 it shall return a value of `-1` and set `errno` to indicate the error.57986 **ERRORS**57987 The `sem_getvalue()` function may fail if:57988 [EINVAL] The `sem` argument does not refer to a valid semaphore.57989 **EXAMPLES**

57990 None.

57991 **APPLICATION USAGE**57992 The `sem_getvalue()` function is part of the Semaphores option and need not be available on all  
57993 implementations.57994 **RATIONALE**

57995 None.

57996 **FUTURE DIRECTIONS**

57997 None.

57998 **SEE ALSO**57999 `semctl()`, `semget()`, `semop()`, `sem_post()`, `sem_timedwait()`, `sem_trywait()`58000 XBD `<semaphore.h>`58001 **CHANGE HISTORY**

58002 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58003 **Issue 6**58004 The `sem_getvalue()` function is marked as part of the Semaphores option.58005 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58006 implementation does not support the Semaphores option.58007 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std  
58008 1003.1d-1999.58009 The `restrict` keyword is added to the `sem_getvalue()` prototype for alignment with the  
58010 ISO/IEC 9899:1999 standard.

58011 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

58012 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS  
58013 section so that the [EINVAL] error becomes optional.

58014 **Issue 7**

58015 The *sem\_getvalue()* function is moved from the Semaphores option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_init()**58016 **NAME**58017 `sem_init` — initialize an unnamed semaphore58018 **SYNOPSIS**

```
58019 #include <semaphore.h>
58020 int sem_init(sem_t *sem, int pshared, unsigned value);
```

58021 **DESCRIPTION**

58022 The `sem_init()` function shall initialize the unnamed semaphore referred to by `sem`. The value of  
 58023 the initialized semaphore shall be `value`. Following a successful call to `sem_init()`, the semaphore  
 58024 may be used in subsequent calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and  
 58025 `sem_destroy()`. This semaphore shall remain usable until the semaphore is destroyed.

58026 If the `pshared` argument has a non-zero value, then the semaphore is shared between processes;  
 58027 in this case, any process that can access the semaphore `sem` can use `sem` for performing  
 58028 `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()` operations.

58029 Only `sem` itself may be used for performing synchronization. The result of referring to copies of  
 58030 `sem` in calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()` is  
 58031 undefined.

58032 If the `pshared` argument is zero, then the semaphore is shared between threads of the process; any  
 58033 thread in this process can use `sem` for performing `sem_wait()`, `sem_timedwait()`, `sem_trywait()`,  
 58034 `sem_post()`, and `sem_destroy()` operations. The use of the semaphore by threads other than those  
 58035 created in the same process is undefined.

58036 Attempting to initialize an already initialized semaphore results in undefined behavior.

58037 **RETURN VALUE**

58038 Upon successful completion, the `sem_init()` function shall initialize the semaphore in `sem` and  
 58039 return 0. Otherwise, it shall return -1 and set `errno` to indicate the error.

58040 **ERRORS**

58041 The `sem_init()` function shall fail if:

- |       |          |   |
|-------|----------|---|
| 58042 | [EINVAL] | The <code>value</code> argument exceeds {SEM_VALUE_MAX}.  |
| 58043 | [ENOSPC] | A resource required to initialize the semaphore has been exhausted, or the<br>58044 limit on semaphores ({SEM_NSEMS_MAX}) has been reached. |
| 58045 | [EPERM]  | The process lacks appropriate privileges to initialize the semaphore.   |

58046 **EXAMPLES**

58047 None.

58048 **APPLICATION USAGE**

58049 The `sem_init()` function is part of the Semaphores option and need not be available on all  
 58050 implementations.

58051 **RATIONALE**

58052 None.

58053 **FUTURE DIRECTIONS**

58054 None.

58055 **SEE ALSO**

58056 [\*sem\\_destroy\(\)\*](#), [\*sem\\_post\(\)\*](#), [\*sem\\_timedwait\(\)\*](#), [\*sem\\_trywait\(\)\*](#)

58057 XBD [\*<semaphore.h>\*](#)

58058 **CHANGE HISTORY**

58059 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58060 **Issue 6**

58061 The *sem\_init()* function is marked as part of the Semaphores option.

58062 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58063 implementation does not support the Semaphores option.

58064 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
58065 1003.1d-1999.

58066 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the  
58067 DESCRIPTION to add the *sem\_timedwait()* function for alignment with IEEE Std 1003.1d-1999.

58068 **Issue 7**

58069 SD5-XSH-ERN-176 is applied.

58070 The *sem\_init()* function is moved from the Semaphores option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_open()**58071 **NAME**58072 `sem_open` — initialize and open a named semaphore58073 **SYNOPSIS**58074 `#include <semaphore.h>`  
58075 `sem_t *sem_open(const char *name, int oflag, ...);`58076 **DESCRIPTION**58077 The `sem_open()` function shall establish a connection between a named semaphore and a process.  
58078 Following a call to `sem_open()` with semaphore name `name`, the process may reference the  
58079 semaphore associated with `name` using the address returned from the call. This semaphore may  
58080 be used in subsequent calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and  
58081 `sem_close()`. The semaphore remains usable by this process until the semaphore is closed by a  
58082 successful call to `sem_close()`, `_exit()`, or one of the `exec` functions.58083 The `oflag` argument controls whether the semaphore is created or merely accessed by the call to  
58084 `sem_open()`. The following flag bits may be set in `oflag`:58085 **O\_CREAT** This flag is used to create a semaphore if it does not already exist. If **O\_CREAT** is  
58086 set and the semaphore already exists, then **O\_CREAT** has no effect, except as noted  
58087 under **O\_EXCL**. Otherwise, `sem_open()` creates a named semaphore. The **O\_CREAT**  
58088 flag requires a third and a fourth argument: `mode`, which is of type **mode\_t**, and  
58089 `value`, which is of type **unsigned**. The semaphore is created with an initial value of  
58090 `value`. Valid initial values for semaphores are less than or equal to  
58091 `{SEM_VALUE_MAX}`.58092 The user ID of the semaphore shall be set to the effective user ID of the process.  
58093 The group ID of the semaphore shall be set to the effective group ID of the process;  
58094 however, if the `name` argument is visible in the file system, the group ID may be set  
58095 to the group ID of the containing directory. The permission bits of the semaphore  
58096 are set to the value of the `mode` argument except those set in the file mode creation  
58097 mask of the process. When bits in `mode` other than the file permission bits are  
58098 specified, the effect is unspecified.58099 After the semaphore named `name` has been created by `sem_open()` with the  
58100 **O\_CREAT** flag, other processes can connect to the semaphore by calling  
58101 `sem_open()` with the same value of `name`.58102 **O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, `sem_open()` fails if the semaphore `name` exists.  
58103 The check for the existence of the semaphore and the creation of the semaphore if it  
58104 does not exist are atomic with respect to other processes executing `sem_open()` with  
58105 **O\_EXCL** and **O\_CREAT** set. If **O\_EXCL** is set and **O\_CREAT** is not set, the effect is  
58106 undefined.58107 If flags other than **O\_CREAT** and **O\_EXCL** are specified in the `oflag` parameter, the  
58108 effect is unspecified.58109 The `name` argument points to a string naming a semaphore object. It is unspecified whether the  
58110 name appears in the file system and is visible to functions that take pathnames as arguments.  
58111 The `name` argument conforms to the construction rules for a pathname, except that the  
58112 interpretation of `<slash>` characters other than the leading `<slash>` character in `name` is  
58113 implementation-defined, and that the length limits for the `name` argument are implementation-  
58114 defined and need not be the same as the pathname limits `{PATH_MAX}` and `{NAME_MAX}`. If  
58115 `name` begins with the `<slash>` character, then processes calling `sem_open()` with the same value of  
58116 `name` shall refer to the same semaphore object, as long as that name has not been removed. If  
58117 `name` does not begin with the `<slash>` character, the effect is implementation-defined.

58118 If a process makes multiple successful calls to *sem\_open()* with the same value for *name*, the same  
 58119 semaphore address shall be returned for each such successful call, provided that there have been  
 58120 no calls to *sem\_unlink()* for this semaphore, and at least one previous successful *sem\_open()* call  
 58121 for this semaphore has not been matched with a *sem\_close()* call.

58122 References to copies of the semaphore produce undefined results.

#### 58123 RETURN VALUE

58124 Upon successful completion, the *sem\_open()* function shall return the address of the semaphore.  
 58125 Otherwise, it shall return a value of SEM\_FAILED and set *errno* to indicate the error. The symbol  
 58126 SEM\_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem\_open()*  
 58127 shall return the value SEM\_FAILED.

#### 58128 ERRORS

58129 If any of the following conditions occur, the *sem\_open()* function shall return SEM\_FAILED and  
 58130 set *errno* to the corresponding value:

58131 [EACCES] The named semaphore exists and the permissions specified by *oflag* are  
 58132 denied, or the named semaphore does not exist and permission to create the  
 58133 named semaphore is denied.

58134 [EEXIST] O\_CREAT and O\_EXCL are set and the named semaphore already exists.

58135 [EINTR] The *sem\_open()* operation was interrupted by a signal.

58136 [EINVAL] The *sem\_open()* operation is not supported for the given name, or O\_CREAT  
 58137 was specified in *oflag* and *value* was greater than {SEM\_VALUE\_MAX}.

58138 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this  
 58139 process.

58140 [ENFILE] Too many semaphores are currently open in the system.

58141 [ENOENT] O\_CREAT is not set and the named semaphore does not exist.

58142 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.

58143 [ENOSPC] There is insufficient space on a storage device for the creation of the new  
 58144 named semaphore.

58145 If any of the following conditions occur, the *sem\_open()* function may return SEM\_FAILED and  
 58146 set *errno* to the corresponding value:

58147 [ENAMETOOLONG]

58148 The length of the *name* argument exceeds {\_POSIX\_PATH\_MAX} on systems  
 58149 XSI that do not support the XSI option or exceeds {\_XOPEN\_PATH\_MAX} on XSI  
 58150 systems, or has a pathname component that is longer than  
 58151 XSI {\_POSIX\_NAME\_MAX} on systems that do not support the XSI option or  
 58152 longer than {\_XOPEN\_NAME\_MAX} on XSI systems.

**sem\_open()**58153 **EXAMPLES**

58154 None.

58155 **APPLICATION USAGE**58156 The *sem\_open()* function is part of the Semaphores option and need not be available on all  
58157 implementations.58158 **RATIONALE**58159 Early drafts required an error return value of `-1` with the type `sem_t *` for the *sem\_open()*  
58160 function, which is not guaranteed to be portable across implementations. The revised text  
58161 provides the symbolic error code `SEM_FAILED` to eliminate the type conflict.58162 **FUTURE DIRECTIONS**58163 A future version might require the *sem\_open()* and *sem\_unlink()* functions to have semantics  
58164 similar to normal file system operations.58165 **SEE ALSO**58166 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*58167 XBD `<semaphore.h>`58168 **CHANGE HISTORY**

58169 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58170 **Issue 6**58171 The *sem\_open()* function is marked as part of the Semaphores option.58172 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an  
58173 implementation does not support the Semaphores option.58174 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
58175 1003.1d-1999.58176 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the  
58177 DESCRIPTION to add the *sem\_timedwait()* function for alignment with IEEE Std 1003.1d-1999.58178 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the  
58179 DESCRIPTION to describe the conditions to return the same semaphore address on a call to  
58180 *sem\_open()*. The words "and at least one previous successful *sem\_open()* call for this semaphore  
58181 has not been matched with a *sem\_close()* call" are added.58182 **Issue 7**58183 Austin Group Interpretation 1003.1-2001 #066 is applied, updating the `[ENOSPC]` error case and  
58184 adding the `[ENOMEM]` error case.58185 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and  
58186 adding `[ENAMETOOLONG]` as a "may fail" error.

58187 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

58188 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
58189 user ID and group ID of the semaphore.58190 The *sem\_open()* function is moved from the Semaphores option to the Base.

58191 **NAME**

58192 sem\_post — unlock a semaphore

58193 **SYNOPSIS**

```
58194 #include <semaphore.h>
58195 int sem_post(sem_t *sem);
```

58196 **DESCRIPTION**

58197 The *sem\_post()* function shall unlock the semaphore referenced by *sem* by performing a  
58198 semaphore unlock operation on that semaphore.

58199 If the semaphore value resulting from this operation is positive, then no threads were blocked  
58200 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

58201 If the value of the semaphore resulting from this operation is zero, then one of the threads  
58202 blocked waiting for the semaphore shall be allowed to return successfully from its call to  
58203 *sem\_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be  
58204 chosen in a manner appropriate to the scheduling policies and parameters in effect for the  
58205 blocked threads. In the case of the schedulers SCHED\_FIFO and SCHED\_RR, the highest  
58206 priority waiting thread shall be unblocked, and if there is more than one highest priority thread  
58207 blocked waiting for the semaphore, then the highest priority thread that has been waiting the  
58208 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread  
58209 to unblock is unspecified.

58210 SS If the Process Sporadic Server option is supported, and the scheduling policy is  
58211 SCHED\_SPORADIC, the semantics are as per SCHED\_FIFO above.

58212 The *sem\_post()* function shall be async-signal-safe and may be invoked from a signal-catching  
58213 function.

58214 **RETURN VALUE**

58215 If successful, the *sem\_post()* function shall return zero; otherwise, the function shall return  $-1$   
58216 and set *errno* to indicate the error.

58217 **ERRORS**

58218 The *sem\_post()* function may fail if:

58219 [EINVAL] The *sem* argument does not refer to a valid semaphore.

58220 **EXAMPLES**

58221 See *sem\_timedwait()*.

58222 **APPLICATION USAGE**

58223 The *sem\_post()* function is part of the Semaphores option and need not be available on all  
58224 implementations.

58225 **RATIONALE**

58226 None.

58227 **FUTURE DIRECTIONS**

58228 None.

58229 **SEE ALSO**

58230 *semctl()*, *semget()*, *semop()*, *sem\_timedwait()*, *sem\_trywait()*

58231 XBD Section 4.11 (on page 110), *<semaphore.h>*

**sem\_post()****CHANGE HISTORY**

58232 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**Issue 6**

58234 The *sem\_post()* function is marked as part of the Semaphores option.

58236 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58237 implementation does not support the Semaphores option.

58238 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
58239 1003.1d-1999.

58240 SCHED\_SPORADIC is added to the list of scheduling policies for which the thread that is to be  
58241 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

58242 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS  
58243 section so that the [EINVAL] error becomes optional.

**Issue 7**

58244 Austin Group Interpretation 1003.1-2001 #156 is applied.

58246 The *sem\_post()* function is moved from the Semaphores option to the Base.

58247 **NAME**58248 `sem_timedwait` — lock a semaphore58249 **SYNOPSIS**

```
58250 #include <semaphore.h>
58251 #include <time.h>

58252 int sem_timedwait(sem_t *restrict sem,
58253                  const struct timespec *restrict abstime);
```

58254 **DESCRIPTION**

58255 The `sem_timedwait()` function shall lock the semaphore referenced by `sem` as in the `sem_wait()`  
 58256 function. However, if the semaphore cannot be locked without waiting for another process or  
 58257 thread to unlock the semaphore by performing a `sem_post()` function, this wait shall be  
 58258 terminated when the specified timeout expires.

58259 The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the  
 58260 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 58261 `abstime`), or if the absolute time specified by `abstime` has already been passed at the time of the  
 58262 call.

58263 The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall  
 58264 be the resolution of the clock on which it is based. The `timespec` data type is defined as a  
 58265 structure in the `<time.h>` header.

58266 Under no circumstance shall the function fail with a timeout if the semaphore can be locked  
 58267 immediately. The validity of the `abstime` need not be checked if the semaphore can be locked  
 58268 immediately.

58269 **RETURN VALUE**

58270 The `sem_timedwait()` function shall return zero if the calling process successfully performed the  
 58271 semaphore lock operation on the semaphore designated by `sem`. If the call was unsuccessful, the  
 58272 state of the semaphore shall be unchanged, and the function shall return a value of `-1` and set  
 58273 `errno` to indicate the error.

58274 **ERRORS**

58275 The `sem_timedwait()` function shall fail if:

58276 [EINVAL] The process or thread would have blocked, and the `abstime` parameter  
 58277 specified a nanoseconds field value less than zero or greater than or equal to  
 58278 1 000 million.

58279 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

58280 The `sem_timedwait()` function may fail if:

58281 [EDEADLK] A deadlock condition was detected.

58282 [EINTR] A signal interrupted this function.

58283 [EINVAL] The `sem` argument does not refer to a valid semaphore.

**sem\_timedwait()****EXAMPLES**

The program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a seconds value that is used to set an alarm timer to generate a SIGALRM signal. This handler performs a `sem_post(3)` to increment the semaphore that is being waited on in `main()` using `sem_timedwait()`. The second command-line argument specifies the length of the timeout, in seconds, for `sem_timedwait()`.

```

58290 #include <unistd.h>
58291 #include <stdio.h>
58292 #include <stdlib.h>
58293 #include <semaphore.h>
58294 #include <time.h>
58295 #include <assert.h>
58296 #include <errno.h>
58297 #include <signal.h>

58298 sem_t sem;

58299 static void
58300 handler(int sig)
58301 {
58302     write(STDOUT_FILENO, "sem_post() from handler\n", 24);
58303     if (sem_post(&sem) == -1) {
58304         write(STDERR_FILENO, "sem_post() failed\n", 18);
58305         _exit(EXIT_FAILURE);
58306     }
58307 }

58308 int
58309 main(int argc, char *argv[])
58310 {
58311     struct sigaction sa;
58312     struct timespec ts;
58313     int s;

58314     if (argc != 3) {
58315         fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
58316             argv[0]);
58317         exit(EXIT_FAILURE);
58318     }

58319     if (sem_init(&sem, 0, 0) == -1) {
58320         perror("sem_init");
58321         exit(EXIT_FAILURE);
58322     }

58323     /* Establish SIGALRM handler; set alarm timer using argv[1] */

58324     sa.sa_handler = handler;
58325     sigemptyset(&sa.sa_mask);
58326     sa.sa_flags = 0;
58327     if (sigaction(SIGALRM, &sa, NULL) == -1) {
58328         perror("sigaction");
58329         exit(EXIT_FAILURE);
58330     }

```

```

58331     alarm(atoi(argv[1]));
58332     /* Calculate relative interval as current time plus
58333        number of seconds given argv[2] */
58334     if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
58335         perror("clock_gettime");
58336         exit(EXIT_FAILURE);
58337     }
58338     ts.tv_sec += atoi(argv[2]);
58339     printf("main() about to call sem_timedwait()\n");
58340     while ((s = sem_timedwait(&sem, &ts)) == -1 && errno == EINTR)
58341         continue; /* Restart if interrupted by handler */
58342     /* Check what happened */
58343     if (s == -1) {
58344         if (errno == ETIMEDOUT)
58345             printf("sem_timedwait() timed out\n");
58346         else
58347             perror("sem_timedwait");
58348     } else
58349         printf("sem_timedwait() succeeded\n");
58350     exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
58351 }

```

**APPLICATION USAGE**

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.285 (on page 79).

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

[sem\\_post\(\)](#), [sem\\_trywait\(\)](#), [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [time\(\)](#)

XBD Section 3.285 (on page 79), [<semaphore.h>](#), [<time.h>](#)

**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

**Issue 7**

The `sem_timedwait()` function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

**sem\_trywait()**58370 **NAME**

58371 sem\_trywait, sem\_wait — lock a semaphore

58372 **SYNOPSIS**

```
58373 #include <semaphore.h>
58374 int sem_trywait(sem_t *sem);
58375 int sem_wait(sem_t *sem);
```

58376 **DESCRIPTION**

58377 The *sem\_trywait()* function shall lock the semaphore referenced by *sem* only if the semaphore is  
 58378 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not  
 58379 lock the semaphore.

58380 The *sem\_wait()* function shall lock the semaphore referenced by *sem* by performing a semaphore  
 58381 lock operation on that semaphore. If the semaphore value is currently zero, then the calling  
 58382 thread shall not return from the call to *sem\_wait()* until it either locks the semaphore or the call is  
 58383 interrupted by a signal.

58384 Upon successful return, the state of the semaphore shall be locked and shall remain locked until  
 58385 the *sem\_post()* function is executed and returns successfully.

58386 The *sem\_wait()* function is interruptible by the delivery of a signal.

58387 **RETURN VALUE**

58388 The *sem\_trywait()* and *sem\_wait()* functions shall return zero if the calling process successfully  
 58389 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was  
 58390 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a  
 58391 value of  $-1$  and set *errno* to indicate the error.

58392 **ERRORS**

58393 The *sem\_trywait()* function shall fail if:

58394 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the  
 58395 *sem\_trywait()* operation.

58396 The *sem\_trywait()* and *sem\_wait()* functions may fail if:

58397 [EDEADLK] A deadlock condition was detected.

58398 [EINTR] A signal interrupted this function.

58399 [EINVAL] The *sem* argument does not refer to a valid semaphore.

58400 **EXAMPLES**

58401 None.

58402 **APPLICATION USAGE**

58403 Applications using these functions may be subject to priority inversion, as discussed in XBD  
 58404 Section 3.285 (on page 79).

58405 The *sem\_trywait()* and *sem\_wait()* functions are part of the Semaphores option and need not be  
 58406 provided on all implementations.

58407 **RATIONALE**

58408 None.

58409 **FUTURE DIRECTIONS**

58410 None.

58411 **SEE ALSO**58412 *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*

58413 XBD Section 3.285 (on page 79), Section 4.11 (on page 110), &lt;semaphore.h&gt;

58414 **CHANGE HISTORY**

58415 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58416 **Issue 6**58417 The *sem\_trywait()* and *sem\_wait()* functions are marked as part of the Semaphores option.58418 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58419 implementation does not support the Semaphores option.58420 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
58421 1003.1d-1999.58422 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS  
58423 section so that the [EINVAL] error becomes optional.58424 **Issue 7**58425 SD5-XSH-ERN-54 is applied, removing the *sem\_wait()* function from the “shall fail” error cases.58426 The *sem\_trywait()* and *sem\_wait()* functions are moved from the Semaphores option to the Base.

**sem\_unlink()**58427 **NAME**58428 `sem_unlink` — remove a named semaphore58429 **SYNOPSIS**58430 `#include <semaphore.h>`  
58431 `int sem_unlink(const char *name);`58432 **DESCRIPTION**58433 The `sem_unlink()` function shall remove the semaphore named by the string *name*. If the  
58434 semaphore named by *name* is currently referenced by other processes, then `sem_unlink()` shall  
58435 have no effect on the state of the semaphore. If one or more processes have the semaphore open  
58436 when `sem_unlink()` is called, destruction of the semaphore is postponed until all references to the  
58437 semaphore have been destroyed by calls to `sem_close()`, `_exit()`, or `exec`. Calls to `sem_open()` to  
58438 recreate or reconnect to the semaphore refer to a new semaphore after `sem_unlink()` is called. The  
58439 `sem_unlink()` call shall not block until all references have been destroyed; it shall return  
58440 immediately.58441 **RETURN VALUE**58442 Upon successful completion, the `sem_unlink()` function shall return a value of 0. Otherwise, the  
58443 semaphore shall not be changed and the function shall return a value of `-1` and set `errno` to  
58444 indicate the error.58445 **ERRORS**58446 The `sem_unlink()` function shall fail if:

58447 [EACCES] Permission is denied to unlink the named semaphore.

58448 [ENOENT] The named semaphore does not exist.

58449 The `sem_unlink()` function may fail if:

58450 [ENAMETOOLONG]

58451 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems  
58452 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI  
58453 systems, or has a pathname component that is longer than  
58454 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or  
58455 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to `sem_unlink()`  
58456 with a *name* argument that contains the same semaphore name as was  
58457 previously used in a successful `sem_open()` call shall not give an  
58458 [ENAMETOOLONG] error.58459 **EXAMPLES**

58460 None.

58461 **APPLICATION USAGE**58462 The `sem_unlink()` function is part of the Semaphores option and need not be available on all  
58463 implementations.58464 **RATIONALE**

58465 None.

58466 **FUTURE DIRECTIONS**58467 A future version might require the `sem_open()` and `sem_unlink()` functions to have semantics  
58468 similar to normal file system operations.

58469 **SEE ALSO**58470 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_open()*58471 XBD <[semaphore.h](#)>58472 **CHANGE HISTORY**

58473 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58474 **Issue 6**58475 The *sem\_unlink()* function is marked as part of the Semaphores option.58476 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58477 implementation does not support the Semaphores option.58478 **Issue 7**58479 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a  
58480 “shall fail” to a “may fail” error.

58481 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

58482 The *sem\_unlink()* function is moved from the Semaphores option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sem\_wait()**58483 **NAME**58484 `sem_wait` — lock a semaphore58485 **SYNOPSIS**58486 `#include <semaphore.h>`58487 `int sem_wait(sem_t *sem);`58488 **DESCRIPTION**58489 Refer to *sem\_trywait()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

58490 **NAME**

58491 semctl — XSI semaphore control operations

58492 **SYNOPSIS**

```
58493 XSI #include <sys/sem.h>
58494 int semctl(int semid, int semnum, int cmd, ...);
```

58495 **DESCRIPTION**

58496 The *semctl()* function operates on XSI semaphores (see XBD Section 4.16, on page 113). It is  
 58497 unspecified whether this function interoperates with the realtime interprocess communication  
 58498 facilities defined in Section 2.8 (on page 497).

58499 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.  
 58500 The fourth argument is optional and depends upon the operation requested. If required, it is of  
 58501 type **union semun**, which the application shall explicitly declare:

```
58502 union semun {
58503     int val;
58504     struct semid_ds *buf;
58505     unsigned short *array;
58506 } arg;
```

58507 The following semaphore control operations as specified by *cmd* are executed with respect to the  
 58508 semaphore specified by *semid* and *semnum*. The level of permission required for each operation  
 58509 is shown with each command; see Section 2.7 (on page 496). The symbolic names for the values  
 58510 of *cmd* are defined in the `<sys/sem.h>` header:

58511	GETVAL	Return the value of <i>semval</i> ; see <code>&lt;sys/sem.h&gt;</code> . Requires read permission.
58512	SETVAL	Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see Section 2.7 (on page 496).
58513		
58514		
58515		
58516	GETPID	Return the value of <i>sempid</i> . Requires read permission.
58517	GETNCNT	Return the value of <i>semncnt</i> . Requires read permission.
58518	GETZCNT	Return the value of <i>semzcnt</i> . Requires read permission.

58519 The following values of *cmd* operate on each *semval* in the set of semaphores:

58520	GETALL	Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.
58521		
58522		
58523	SETALL	Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.
58524		
58525		
58526		
58527		

58528 The following values of *cmd* are also available:

58529	IPC_STAT	Place the current value of each member of the <b>semid_ds</b> data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in <code>&lt;sys/sem.h&gt;</code> . Requires read permission.
58530		
58531		
58532		

**semctl()**

58533	IPC_SET	Set the value of the following members of the <b>semid_ds</b> data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
58534		
58535		
58536		<i>sem_perm.uid</i>
58537		<i>sem_perm.gid</i>
58538		<i>sem_perm.mode</i>
58539		The mode bits specified in <a href="#">Section 2.7.1</a> (on page 496) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.
58540		
58541		
58542		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
58543		
58544		
58545		
58546	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and <b>semid_ds</b> data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
58547		
58548		
58549		
58550		
58551		
58552	<b>RETURN VALUE</b>	
58553		If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:
58554	GETVAL	The value of <i>semval</i> .
58555	GETPID	The value of <i>sempid</i> .
58556	GETNCNT	The value of <i>semncnt</i> .
58557	GETZCNT	The value of <i>semzcnt</i> .
58558	All others	0.
58559		Otherwise, <i>semctl()</i> shall return $-1$ and set <i>errno</i> to indicate the error.
58560	<b>ERRORS</b>	
58561		The <i>semctl()</i> function shall fail if:
58562	[EACCES]	Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 496).
58563		
58564	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
58565		
58566		
58567	[EPERM]	The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .
58568		
58569		
58570		
58571	[ERANGE]	The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.
58572		

58573 **EXAMPLES**

58574 None.

58575 **APPLICATION USAGE**

58576 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a  
 58577 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for  
 58578 backwards-compatibility.

58579 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 58580 Application developers who need to use IPC should design their applications so that modules  
 58581 using the IPC routines described in Section 2.7 (on page 496) can be easily modified to use the  
 58582 alternative interfaces.

58583 **RATIONALE**

58584 None.

58585 **FUTURE DIRECTIONS**

58586 None.

58587 **SEE ALSO**

58588 Section 2.7 (on page 496), Section 2.8 (on page 497), *semget()*, *semop()*, *sem\_close()*, *sem\_destroy()*,  
 58589 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_trywait()*, *sem\_unlink()*

58590 XBD Section 4.16 (on page 113), [<sys/sem.h>](#)58591 **CHANGE HISTORY**

58592 First released in Issue 2. Derived from Issue 2 of the SVID.

58593 **Issue 5**

58594 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 58595 DIRECTIONS to the APPLICATION USAGE section.

**semget()**58596 **NAME**58597 `semget` — get set of XSI semaphores58598 **SYNOPSIS**

```
58599 XSI #include <sys/sem.h>
58600 int semget(key_t key, int nsems, int semflg);
```

58601 **DESCRIPTION**

58602 The `semget()` function operates on XSI semaphores (see XBD Section 4.16, on page 113). It is  
 58603 unspecified whether this function interoperates with the realtime interprocess communication  
 58604 facilities defined in Section 2.8 (on page 497).

58605 The `semget()` function shall return the semaphore identifier associated with `key`.

58606 A semaphore identifier with its associated `semid_ds` data structure and its associated set of  
 58607 `nsems` semaphores (see `<sys/sem.h>`) is created for `key` if one of the following is true:

- 58608 • The argument `key` is equal to `IPC_PRIVATE`.
- 58609 • The argument `key` does not already have a semaphore identifier associated with it and  
 58610 (`semflg & IPC_CREAT`) is non-zero.

58611 Upon creation, the `semid_ds` data structure associated with the new semaphore identifier is  
 58612 initialized as follows:

- 58613 • In the operation permissions structure `sem_perm.cuid`, `sem_perm.uid`, `sem_perm.cgid`, and  
 58614 `sem_perm.gid` shall be set equal to the effective user ID and effective group ID, respectively,  
 58615 of the calling process.
- 58616 • The low-order 9 bits of `sem_perm.mode` shall be set equal to the low-order 9 bits of `semflg`.
- 58617 • The variable `sem_nsems` shall be set equal to the value of `nsems`.
- 58618 • The variable `sem_otime` shall be set equal to 0 and `sem_ctime` shall be set equal to the current  
 58619 time.
- 58620 • The data structure associated with each semaphore in the set need not be initialized. The  
 58621 `semctl()` function with the command `SETVAL` or `SETALL` can be used to initialize each  
 58622 semaphore.

58623 **RETURN VALUE**

58624 Upon successful completion, `semget()` shall return a non-negative integer, namely a semaphore  
 58625 identifier; otherwise, it shall return `-1` and set `errno` to indicate the error.

58626 **ERRORS**

58627 The `semget()` function shall fail if:

- |       |          |   |
|-------|----------|---|
| 58628 | [EACCES] | A semaphore identifier exists for <code>key</code> , but operation permission as specified by the low-order 9 bits of <code>semflg</code> would not be granted; see Section 2.7 (on page 496).  |
| 58629 |          |   |
| 58630 |          |   |
| 58631 | [EEXIST] | A semaphore identifier exists for the argument <code>key</code> but <code>((semflg &amp; IPC_CREAT) &amp;&amp; (semflg &amp; IPC_EXCL))</code> is non-zero.   |
| 58632 |          |   |
| 58633 | [EINVAL] | The value of <code>nsems</code> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <code>key</code> , but the number of semaphores in the set associated with it is less than <code>nsems</code> and <code>nsems</code> is not equal to 0. |
| 58634 |          |   |
| 58635 |          |   |
| 58636 |          |   |

58637	[ENOENT]	A semaphore identifier does not exist for the argument <i>key</i> and ( <i>semflg</i> &IPC_CREAT) is equal to 0.
58638		
58639	[ENOSPC]	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.
58640		

58641 **EXAMPLES**58642 **Creating a Semaphore Identifier**

58643 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 58644 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 58645 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 58646 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
 58647 program attempts to create one semaphore with read/write permission for all. It also uses the  
 58648 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

58649 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 58650 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 58651 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 58652 the program.

```

58653 #include <sys/types.h>
58654 #include <stdio.h>
58655 #include <sys/ipc.h>
58656 #include <sys/sem.h>
58657 #include <sys/stat.h>
58658 #include <errno.h>
58659 #include <unistd.h>
58660 #include <stdlib.h>
58661 #include <pwd.h>
58662 #include <fcntl.h>
58663 #include <limits.h>
58664 ...
58665 key_t semkey;
58666 int semid, pfd, fw;
58667 struct sembuf sbuf;
58668 char *lgn;
58669 char filename[PATH_MAX+1];
58670 struct stat outstat;
58671 struct passwd *pw;
58672 ...
58673 /* Get unique key for semaphore. */
58674 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
58675     perror("IPC error: ftok"); exit(1);
58676 }
58677
58678 /* Get semaphore ID associated with this key. */
58679 if ((semid = semget(semkey, 0, 0)) == -1) {
58680     /* Semaphore does not exist - Create. */
58681     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
58682         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
58683     {
58684         /* Initialize the semaphore. */

```

**semget()**

```

58684         sbuf.sem_num = 0;
58685         sbuf.sem_op = 2; /* This is the number of runs
58686                        without queuing. */
58687         sbuf.sem_flg = 0;
58688         if (semop(semid, &sbuf, 1) == -1) {
58689             perror("IPC error: semop"); exit(1);
58690         }
58691     }
58692     else if (errno == EEXIST) {
58693         if ((semid = semget(semkey, 0, 0)) == -1) {
58694             perror("IPC error 1: semget"); exit(1);
58695         }
58696     }
58697     else {
58698         perror("IPC error 2: semget"); exit(1);
58699     }
58700 }
58701 ...

```

**58702 APPLICATION USAGE**

58703 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
58704 Application developers who need to use IPC should design their applications so that modules  
58705 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the  
58706 alternative interfaces.

**58707 RATIONALE**

58708 None.

**58709 FUTURE DIRECTIONS**

58710 None.

**58711 SEE ALSO**

58712 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [semctl\(\)](#), [semop\(\)](#), [sem\\_close\(\)](#), [sem\\_destroy\(\)](#),  
58713 [sem\\_getvalue\(\)](#), [sem\\_init\(\)](#), [sem\\_open\(\)](#), [sem\\_post\(\)](#), [sem\\_trywait\(\)](#), [sem\\_unlink\(\)](#)

58714 XBD [Section 4.16](#) (on page 113), [<sys/sem.h>](#)

**58715 CHANGE HISTORY**

58716 First released in Issue 2. Derived from Issue 2 of the SVID.

**58717 Issue 5**

58718 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
58719 DIRECTIONS to a new APPLICATION USAGE section.

**58720 Issue 6**

58721 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the  
58722 DESCRIPTION from "each semaphore in the set shall not be initialized" to "each semaphore in  
58723 the set need not be initialized".

58724 **NAME**

58725 semop — XSI semaphore operations

58726 **SYNOPSIS**

```
58727 XSI #include <sys/sem.h>
58728 int semop(int semid, struct sembuf *sops, size_t nsops);
```

58729 **DESCRIPTION**

58730 The *semop()* function operates on XSI semaphores (see XBD Section 4.16, on page 113). It is  
 58731 unspecified whether this function interoperates with the realtime interprocess communication  
 58732 facilities defined in Section 2.8 (on page 497).

58733 The *semop()* function shall perform atomically a user-defined array of semaphore operations in  
 58734 array order on the set of semaphores associated with the semaphore identifier specified by the  
 58735 argument *semid*.

58736 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The  
 58737 implementation shall not modify elements of this array unless the application uses  
 58738 implementation-defined extensions.

58739 The argument *nsops* is the number of such structures in the array.

58740 Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

58745 Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore  
 58746 specified by *semid* and *sem\_num*.

58747 The variable *sem\_op* specifies one of three semaphore operations:

- 58748 1. If *sem\_op* is a negative integer and the calling process has alter permission, one of the  
 58749 following shall occur:
  - 58750 • If *semval* (see <sys/sem.h>) is greater than or equal to the absolute value of *sem\_op*,  
 58751 the absolute value of *sem\_op* is subtracted from *semval*. Also, if (*sem\_flg*  
 58752 &SEM\_UNDO) is non-zero, the absolute value of *sem\_op* shall be added to the  
 58753 *semadj* value of the calling process for the specified semaphore.
  - 58754 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is  
 58755 non-zero, *semop()* shall return immediately.
  - 58756 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is 0,  
 58757 *semop()* shall increment the *semncnt* associated with the specified semaphore and  
 58758 suspend execution of the calling thread until one of the following conditions occurs:
    - 58759 — The value of *semval* becomes greater than or equal to the absolute value of  
 58760 *sem\_op*. When this occurs, the value of *semncnt* associated with the specified  
 58761 semaphore shall be decremented, the absolute value of *sem\_op* shall be  
 58762 subtracted from *semval* and, if (*sem\_flg* &SEM\_UNDO) is non-zero, the  
 58763 absolute value of *sem\_op* shall be added to the *semadj* value of the calling  
 58764 process for the specified semaphore.

**semop()**

- 58765 — The *semid* for which the calling thread is awaiting action is removed from the  
58766 system. When this occurs, *errno* shall be set equal to [EIDRM] and  $-1$  shall be  
58767 returned.
- 58768 — The calling thread receives a signal that is to be caught. When this occurs, the  
58769 value of *semncnt* associated with the specified semaphore shall be  
58770 decremented, and the calling thread shall resume execution in the manner  
58771 prescribed in *sigaction()*.
- 58772 2. If *sem\_op* is a positive integer and the calling process has alter permission, the value of  
58773 *sem\_op* shall be added to *semval* and, if (*sem\_flg* & SEM\_UNDO) is non-zero, the value of  
58774 *sem\_op* shall be subtracted from the *semadj* value of the calling process for the specified  
58775 semaphore.
- 58776 3. If *sem\_op* is 0 and the calling process has read permission, one of the following shall occur:
- 58777 • If *semval* is 0, *semop()* shall return immediately.
  - 58778 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is non-zero, *semop()* shall return  
58779 immediately.
  - 58780 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is 0, *semop()* shall increment the  
58781 *semzcnt* associated with the specified semaphore and suspend execution of the  
58782 calling thread until one of the following occurs:
    - 58783 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated  
58784 with the specified semaphore shall be decremented.
    - 58785 — The *semid* for which the calling thread is awaiting action is removed from the  
58786 system. When this occurs, *errno* shall be set equal to [EIDRM] and  $-1$  shall be  
58787 returned.
    - 58788 — The calling thread receives a signal that is to be caught. When this occurs, the  
58789 value of *semzcnt* associated with the specified semaphore shall be  
58790 decremented, and the calling thread shall resume execution in the manner  
58791 prescribed in *sigaction()*.
- 58792 Upon successful completion, the value of *sempid* for each semaphore specified in the array  
58793 pointed to by *sops* shall be set equal to the process ID of the calling process.

**RETURN VALUE**

58794 Upon successful completion, *semop()* shall return 0; otherwise, it shall return  $-1$  and set *errno* to  
58795 indicate the error.  
58796

**ERRORS**

58797 The *semop()* function shall fail if:  
58798

- |       |          |   |
|-------|----------|---|
| 58799 | [E2BIG]  | The value of <i>nsops</i> is greater than the system-imposed maximum.   |
| 58800 | [EACCES] | Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 496).                                     |
| 58801 |          |   |
| 58802 | [EAGAIN] | The operation would result in suspension of the calling process but ( <i>sem_flg</i> & IPC_NOWAIT) is non-zero.                           |
| 58803 |          |   |
| 58804 | [EFBIG]  | The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i> . |
| 58805 |          |   |

58806	[EIDRM]	The semaphore identifier <i>semid</i> is removed from the system.
58807	[EINTR]	The <i>semop()</i> function was interrupted by a signal.
58808	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.
58809		
58810		
58811	[ENOSPC]	The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.
58812		
58813	[ERANGE]	An operation would cause a <i>semval</i> to overflow the system-imposed limit, or an operation would cause a <i>semadj</i> value to overflow the system-imposed limit.
58814		
58815		

58816 **EXAMPLES**58817 **Setting Values in Semaphores**

58818 The following example sets the values of the two semaphores associated with the *semid* identifier  
58819 to the values contained in the *sb* array.

```
58820 #include <sys/sem.h>
58821 ...
58822 int semid;
58823 struct sembuf sb[2];
58824 int nsops = 2;
58825 int result;

58826 /* Adjust value of semaphore in the semaphore array semid. */
58827 sb[0].sem_num = 0;
58828 sb[0].sem_op = -1;
58829 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
58830 sb[1].sem_num = 1;
58831 sb[1].sem_op = 1;
58832 sb[1].sem_flg = 0;

58833 result = semop(semid, sb, nsops);
```

58834 **Creating a Semaphore Identifier**

58835 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
58836 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
58837 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
58838 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
58839 program attempts to create one semaphore with read/write permission for all. It also uses the  
58840 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

58841 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
58842 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
58843 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
58844 the program.

58845 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM\_UNDO  
58846 option releases the semaphore when the process exits, waiting until there are less than two  
58847 processes running concurrently.

## semop()

```

58848     #include <sys/types.h>
58849     #include <stdio.h>
58850     #include <sys/ipc.h>
58851     #include <sys/sem.h>
58852     #include <sys/stat.h>
58853     #include <errno.h>
58854     #include <unistd.h>
58855     #include <stdlib.h>
58856     #include <pwd.h>
58857     #include <fcntl.h>
58858     #include <limits.h>
58859     ...
58860     key_t semkey;
58861     int semid, pfd, fv;
58862     struct sembuf sbuf;
58863     char *lgn;
58864     char filename[PATH_MAX+1];
58865     struct stat outstat;
58866     struct passwd *pw;
58867     ...
58868     /* Get unique key for semaphore. */
58869     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
58870         perror("IPC error: ftok"); exit(1);
58871     }
58872     /* Get semaphore ID associated with this key. */
58873     if ((semid = semget(semkey, 0, 0)) == -1) {
58874         /* Semaphore does not exist - Create. */
58875         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
58876             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
58877         {
58878             /* Initialize the semaphore. */
58879             sbuf.sem_num = 0;
58880             sbuf.sem_op = 2; /* This is the number of runs without queuing. */
58881             sbuf.sem_flg = 0;
58882             if (semop(semid, &sbuf, 1) == -1) {
58883                 perror("IPC error: semop"); exit(1);
58884             }
58885         }
58886         else if (errno == EEXIST) {
58887             if ((semid = semget(semkey, 0, 0)) == -1) {
58888                 perror("IPC error 1: semget"); exit(1);
58889             }
58890         }
58891         else {
58892             perror("IPC error 2: semget"); exit(1);
58893         }
58894     }
58895     ...
58896     sbuf.sem_num = 0;
58897     sbuf.sem_op = -1;
58898     sbuf.sem_flg = SEM_UNDO;

```

```

58899     if (semop(semid, &sbuf, 1) == -1) {
58900         perror("IPC Error: semop"); exit(1);
58901     }

```

**58902 APPLICATION USAGE**

58903 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 58904 Application developers who need to use IPC should design their applications so that modules  
 58905 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the  
 58906 alternative interfaces.

**58907 RATIONALE**

58908 None.

**58909 FUTURE DIRECTIONS**

58910 None.

**58911 SEE ALSO**

58912 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*,  
 58913 *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_trywait()*,  
 58914 *sem\_unlink()*

58915 XBD [Section 4.16](#) (on page 113), [<sys/ipc.h>](#), [<sys/sem.h>](#), [<sys/types.h>](#)

**58916 CHANGE HISTORY**

58917 First released in Issue 2. Derived from Issue 2 of the SVID.

**58918 Issue 5**

58919 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 58920 DIRECTIONS to a new APPLICATION USAGE section.

**58921 Issue 7**

58922 SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the  
 58923 operations in *sops* will be performed when there are multiple operations.

**send()**58924 **NAME**58925 `send` — send a message on a socket58926 **SYNOPSIS**58927 `#include <sys/socket.h>`58928 `ssize_t send(int socket, const void *buffer, size_t length, int flags);`58929 **DESCRIPTION**

58930 The `send()` function shall initiate transmission of a message from the specified socket to its peer.  
 58931 The `send()` function shall send a message only when the socket is connected. If the socket is a  
 58932 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

58933 The `send()` function takes the following arguments:

58934	<i>socket</i>	Specifies the socket file descriptor.
58935	<i>buffer</i>	Points to the buffer containing the message to send.
58936	<i>length</i>	Specifies the length of the message in bytes.
58937	<i>flags</i>	Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:
58939	MSG_EOR	Terminates a record (if supported by the protocol).
58940	MSG_OOB	Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.
58941		
58942		
58943	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.
58944		
58945		
58946		

58947 The length of the message to be sent is specified by the *length* argument. If the message is too  
 58948 long to pass through the underlying protocol, `send()` shall fail and no data shall be transmitted.

58949 Successful completion of a call to `send()` does not guarantee delivery of the message. A return  
 58950 value of `-1` indicates only locally-detected errors.

58951 If space is not available at the sending socket to hold the message to be transmitted, and the  
 58952 socket file descriptor does not have `O_NONBLOCK` set, `send()` shall block until space is  
 58953 available. If space is not available at the sending socket to hold the message to be transmitted,  
 58954 and the socket file descriptor does have `O_NONBLOCK` set, `send()` shall fail. The `select()` and  
 58955 `poll()` functions can be used to determine when it is possible to send more data.

58956 The socket in use may require the process to have appropriate privileges to use the `send()`  
 58957 function.

58958 **RETURN VALUE**

58959 Upon successful completion, `send()` shall return the number of bytes sent. Otherwise, `-1` shall be  
 58960 returned and `errno` set to indicate the error.

58961 **ERRORS**58962 The `send()` function shall fail if:

58963 [EAGAIN] or [EWOULDBLOCK]

58964 The socket's file descriptor is marked `O_NONBLOCK` and the requested  
 58965 operation would block.

58966	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
58967	[ECONNRESET]	A connection was forcibly closed by a peer.
58968	[EDESTADDRREQ]	
58969		The socket is not connection-mode and no peer address is set.
58970	[EINTR]	A signal interrupted <i>send()</i> before any data was transmitted.
58971	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
58972	[ENOTCONN]	The socket is not connected.
58973	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
58974	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
58975		
58976	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.
58977		
58978		
58979		
58980		The <i>send()</i> function may fail if:
58981	[EACCES]	The calling process does not have appropriate privileges.
58982	[EIO]	An I/O error occurred while reading from or writing to the file system.
58983	[ENETDOWN]	The local network interface used to reach the destination is down.
58984	[ENETUNREACH]	
58985		No route to the network is present.
58986	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
58987	<b>EXAMPLES</b>	
58988		None.
58989	<b>APPLICATION USAGE</b>	
58990		The <i>send()</i> function is equivalent to <i>sendto()</i> with a null pointer <i>dest_len</i> argument, and to <i>write()</i> if no flags are used.
58991		
58992	<b>RATIONALE</b>	
58993		None.
58994	<b>FUTURE DIRECTIONS</b>	
58995		None.
58996	<b>SEE ALSO</b>	
58997		<a href="#">connect()</a> , <a href="#">getsockopt()</a> , <a href="#">poll()</a> , <a href="#">pselect()</a> , <a href="#">recv()</a> , <a href="#">recvfrom()</a> , <a href="#">recvmsg()</a> , <a href="#">sendmsg()</a> , <a href="#">sendto()</a> , <a href="#">setsockopt()</a> , <a href="#">shutdown()</a> , <a href="#">socket()</a>
58998		
58999		XBD <a href="#">&lt;sys/socket.h&gt;</a>
59000	<b>CHANGE HISTORY</b>	
59001		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
59002	<b>Issue 7</b>	
59003		Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify the behavior when the socket is a connectionless-mode socket.
59004		

**send()**

- 59005 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 59006
- 59007 The [EPIPE] error is modified.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59008 **NAME**

59009 sendmsg — send a message on a socket using a message structure

59010 **SYNOPSIS**

59011 #include &lt;sys/socket.h&gt;

59012 ssize\_t sendmsg(int *socket*, const struct msghdr \**message*, int *flags*);59013 **DESCRIPTION**

59014 The *sendmsg()* function shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified in **msghdr** (overriding the pre-specified peer address), or the function shall return  $-1$  and set *errno* to [EISCONN]. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

59021 The *sendmsg()* function takes the following arguments:59022 *socket* Specifies the socket file descriptor.

59023 *message* Points to a **msghdr** structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The *msg\_flags* member is ignored.

59026 *flags* Specifies the type of message transmission. The application may specify 0 or the following flag:

59028 MSG\_EOR Terminates a record (if supported by the protocol).

59029 MSG\_OOB Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific.

59032 MSG\_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.

59036 The *msg\_iov* and *msg\_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg\_iov* is sent in turn.

59041 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A return value of  $-1$  indicates only locally-detected errors.

59043 If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O\_NONBLOCK set, the *sendmsg()* function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O\_NONBLOCK set, the *sendmsg()* function shall fail.

59048 If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg()* shall fail if the SO\_BROADCAST option is not set for the socket.

59050 The socket in use may require the process to have appropriate privileges to use the *sendmsg()* function.

**sendmsg()**59052 **RETURN VALUE**

59053 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, -1  
 59054 shall be returned and *errno* set to indicate the error.

59055 **ERRORS**

59056 The *sendmsg()* function shall fail if:

59057 [EAGAIN] or [EWOULDBLOCK]

59058 The socket's file descriptor is marked O\_NONBLOCK and the requested  
 59059 operation would block.

59060 [EAFNOSUPPORT]

59061 Addresses in the specified address family cannot be used with this socket.

59062 [EBADF] The *socket* argument is not a valid file descriptor.

59063 [ECONNRESET] A connection was forcibly closed by a peer.

59064 [EINTR] A signal interrupted *sendmsg()* before any data was transmitted.

59065 [EINVAL] The sum of the *iov\_len* values overflows an **ssize\_t**.

59066 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the  
 59067 *msg\_iovlen* member of the **msghdr** structure pointed to by *message* is less than  
 59068 or equal to 0 or is greater than {IOV\_MAX}.

59069 [ENOTCONN] The socket is connection-mode but is not connected.

59070 [ENOTSOCK] The *socket* argument does not refer to a socket.

59071 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
 59072 more of the values set in *flags*.

59073 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
 59074 no longer connected. In the latter case, and if the socket is of type  
 59075 SOCK\_STREAM or SOCK\_SEQPACKET and the MSG\_NOSIGNAL flag is not  
 59076 set, the SIGPIPE signal is generated to the calling thread.

59077 If the address family of the socket is AF\_UNIX, then *sendmsg()* shall fail if:

59078 [EIO] An I/O error occurred while reading from or writing to the file system.

59079 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
 59080 in the socket address.

59081 [ENAMETOOLONG]

59082 The length of a component of a pathname is longer than {NAME\_MAX}.

59083 [ENOENT]

59084 A component of the pathname does not name an existing file or the path name  
 59084 is an empty string.

59085 [ENOTDIR]

59086 A component of the path prefix of the pathname in the socket address is not a  
 59087 directory, or the pathname in the socket address contains at least one  
 59088 non-*<slash>* character and ends with one or more trailing *<slash>* characters  
 59089 and the last pathname component names an existing file that is neither a  
 59089 directory nor a symbolic link to a directory.

59090 The *sendmsg()* function may fail if:

59091 [EACCES]

59092 Search permission is denied for a component of the path prefix; or write access  
 59092 to the named socket is denied.

59093	[EDESTADDRREQ]	
59094		The socket is not connection-mode and does not have its peer address set, and
59095		no destination address was specified.
59096	[EHOSTUNREACH]	
59097		The destination host cannot be reached (probably because the host is down or
59098		a remote router cannot reach it).
59099	[EIO]	An I/O error occurred while reading from or writing to the file system.
59100	[EISCONN]	A destination address was specified and the socket is already connected.
59101	[ENETDOWN]	The local network interface used to reach the destination is down.
59102	[ENETUNREACH]	
59103		No route to the network is present.
59104	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
59105	[ENOMEM]	Insufficient memory was available to fulfill the request.
59106		If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> may fail if:
59107	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
59108		resolution of the pathname in the socket address.
59109	[ENAMETOOLONG]	
59110		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
59111		symbolic link produced an intermediate result with a length that exceeds
59112		{PATH_MAX}.

**59113 EXAMPLES**

59114 Done.

**59115 APPLICATION USAGE**

59116 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

**59117 RATIONALE**

59118 None.

**59119 FUTURE DIRECTIONS**

59120 None.

**59121 SEE ALSO**

59122 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*,  
59123 *shutdown()*, *socket()*

59124 XBD <[sys/socket.h](#)>

**59125 CHANGE HISTORY**

59126 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

59127 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
59128 [ELOOP] error condition is added.

**59129 Issue 7**

59130 Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.

59131 Austin Group Interpretation 1003.1-2001 #143 is applied.

59132 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended  
59133 API Set Part 2.

**sendmsg()**

59134

The [EPIPE] error is modified.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59135 **NAME**59136 `sendto` — send a message on a socket59137 **SYNOPSIS**59138 `#include <sys/socket.h>`

```
59139     ssize_t sendto(int socket, const void *message, size_t length,
59140                   int flags, const struct sockaddr *dest_addr,
59141                   socklen_t dest_len);
```

59142 **DESCRIPTION**

59143 The `sendto()` function shall send a message through a connection-mode or connectionless-mode  
 59144 socket.

59145 If the socket is a connectionless-mode socket, the message shall be sent to the address specified  
 59146 by `dest_addr` if no pre-specified peer address has been set. If a peer address has been pre-  
 59147 specified, either the message shall be sent to the address specified by `dest_addr` (overriding the  
 59148 pre-specified peer address), or the function shall return `-1` and set `errno` to `[EISCONN]`.

59149 If the socket is connection-mode, `dest_addr` shall be ignored.

59150 The `sendto()` function takes the following arguments:

59151 `socket` Specifies the socket file descriptor.

59152 `message` Points to a buffer containing the message to be sent.

59153 `length` Specifies the size of the message in bytes.

59154 `flags` Specifies the type of message transmission. Values of this argument are  
 59155 formed by logically OR'ing zero or more of the following flags:

59156 `MSG_EOR` Terminates a record (if supported by the protocol).

59157 `MSG_OOB` Sends out-of-band data on sockets that support out-of-  
 59158 band data. The significance and semantics of out-of-  
 59159 band data are protocol-specific.

59160 `MSG_NOSIGNAL` Requests not to send the SIGPIPE signal if an attempt to  
 59161 send is made on a stream-oriented socket that is no  
 59162 longer connected. The `[EPIPE]` error shall still be  
 59163 returned.

59164 `dest_addr` Points to a `sockaddr` structure containing the destination address. The length  
 59165 and format of the address depend on the address family of the socket.

59166 `dest_len` Specifies the length of the `sockaddr` structure pointed to by the `dest_addr`  
 59167 argument.

59168 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 59169 socket protocol, `sendto()` shall fail if the `SO_BROADCAST` option is not set for the socket.

59170 The `dest_addr` argument specifies the address of the target.

59171 The `length` argument specifies the length of the message.

59172 Successful completion of a call to `sendto()` does not guarantee delivery of the message. A return  
 59173 value of `-1` indicates only locally-detected errors.

59174 If space is not available at the sending socket to hold the message to be transmitted and the  
 59175 socket file descriptor does not have `O_NONBLOCK` set, `sendto()` shall block until space is  
 59176 available. If space is not available at the sending socket to hold the message to be transmitted

**sendto()**

- 59177 and the socket file descriptor does have O\_NONBLOCK set, *sendto()* shall fail.
- 59178 The socket in use may require the process to have appropriate privileges to use the *sendto()*  
59179 function.
- 59180 **RETURN VALUE**
- 59181 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise, -1 shall  
59182 be returned and *errno* set to indicate the error.
- 59183 **ERRORS**
- 59184 The *sendto()* function shall fail if:
- 59185 [EAFNOSUPPORT]  
59186 Addresses in the specified address family cannot be used with this socket.
- 59187 [EAGAIN] or [EWOULDBLOCK]  
59188 The socket's file descriptor is marked O\_NONBLOCK and the requested  
59189 operation would block.
- 59190 [EBADF] The *socket* argument is not a valid file descriptor.
- 59191 [ECONNRESET] A connection was forcibly closed by a peer.
- 59192 [EINTR] A signal interrupted *sendto()* before any data was transmitted.
- 59193 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 59194 [ENOTCONN] The socket is connection-mode but is not connected.
- 59195 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 59196 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
59197 more of the values set in *flags*.
- 59198 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
59199 no longer connected. In the latter case, and if the socket is of type  
59200 SOCK\_STREAM or SOCK\_SEQPACKET and the MSG\_NOSIGNAL flag is not  
59201 set, the SIGPIPE signal is generated to the calling thread.
- 59202 If the address family of the socket is AF\_UNIX, then *sendto()* shall fail if:
- 59203 [EIO] An I/O error occurred while reading from or writing to the file system.
- 59204 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
59205 in the socket address.
- 59206 [ENAMETOOLONG]  
59207 The length of a component of a pathname is longer than {NAME\_MAX}.
- 59208 [ENOENT] A component of the pathname does not name an existing file or the pathname  
59209 is an empty string.
- 59210 [ENOTDIR] A component of the path prefix of the pathname in the socket address is not a  
59211 directory, or the pathname in the socket address contains at least one  
59212 non-`<slash>` character and ends with one or more trailing `<slash>` characters  
59213 and the last pathname component names an existing file that is neither a  
59214 directory nor a symbolic link to a directory.

- 59215 The *sendto()* function may fail if:
- 59216 [EACCES] Search permission is denied for a component of the path prefix; or write access  
59217 to the named socket is denied.
- 59218 [EDESTADDRREQ]  
59219 The socket is not connection-mode and does not have its peer address set, and  
59220 no destination address was specified.
- 59221 [EHOSTUNREACH]  
59222 The destination host cannot be reached (probably because the host is down or  
59223 a remote router cannot reach it).
- 59224 [EINVAL] The *dest\_len* argument is not a valid length for the address family.
- 59225 [EIO] An I/O error occurred while reading from or writing to the file system.
- 59226 [EISCONN] A destination address was specified and the socket is already connected.
- 59227 [ENETDOWN] The local network interface used to reach the destination is down.
- 59228 [ENETUNREACH]  
59229 No route to the network is present.
- 59230 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 59231 [ENOMEM] Insufficient memory was available to fulfill the request.
- 59232 If the address family of the socket is AF\_UNIX, then *sendto()* may fail if:
- 59233 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
59234 resolution of the pathname in the socket address.
- 59235 [ENAMETOOLONG]  
59236 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
59237 symbolic link produced an intermediate result with a length that exceeds  
59238 {PATH\_MAX}.
- 59239 **EXAMPLES**  
59240 None.
- 59241 **APPLICATION USAGE**  
59242 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 59243 **RATIONALE**  
59244 None.
- 59245 **FUTURE DIRECTIONS**  
59246 None.
- 59247 **SEE ALSO**  
59248 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *setsockopt()*,  
59249 *shutdown()*, *socket()*
- 59250 XBD <[sys/socket.h](#)>
- 59251 **CHANGE HISTORY**  
59252 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 59253 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
59254 [ELOOP] error condition is added.

**sendto()***System Interfaces*59255 **Issue 7**

59256 Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN]  
59257 error and the DESCRIPTION.

59258 Austin Group Interpretation 1003.1-2001 #143 is applied, clarifying the [ENAMETOOLONG]  
59259 error condition.

59260 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended  
59261 API Set Part 2.

59262 The [EPIPE] error is modified.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59263 **NAME**

59264 setbuf — assign buffering to a stream

59265 **SYNOPSIS**

59266 #include &lt;stdio.h&gt;

59267 void setbuf(FILE \*restrict stream, char \*restrict buf);

59268 **DESCRIPTION**

59269 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 59270 conflict between the requirements described here and the ISO C standard is unintentional. This  
 59271 volume of POSIX.1-2008 defers to the ISO C standard.

59272 Except that it returns no value, the function call:

59273 setbuf(stream, buf)

59274 shall be equivalent to:

59275 setvbuf(stream, buf, \_IOFBF, BUFSIZ)

59276 if *buf* is not a null pointer, or to:

59277 setvbuf(stream, buf, \_IONBF, BUFSIZ)

59278 if *buf* is a null pointer.59279 **RETURN VALUE**59280 The *setbuf()* function shall not return a value.59281 **ERRORS**

59282 No errors are defined.

59283 **EXAMPLES**

59284 None.

59285 **APPLICATION USAGE**

59286 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
 59287 and then failing to close the stream in the same block.

59288 With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ  
 59289 bytes are used for the buffer area.

59290 **RATIONALE**

59291 None.

59292 **FUTURE DIRECTIONS**

59293 None.

59294 **SEE ALSO**59295 *fcntl()*, *setvbuf()*

59296 XBD &lt;stdio.h&gt;

59297 **CHANGE HISTORY**

59298 First released in Issue 1. Derived from Issue 1 of the SVID.

59299 **Issue 6**59300 The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

**setegid()**59301 **NAME**

59302           setegid — set the effective group ID

59303 **SYNOPSIS**

59304           #include &lt;unistd.h&gt;

59305           int setegid(gid\_t gid);

59306 **DESCRIPTION**59307           If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate  
59308 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group  
59309 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.59310           The *setegid()* function shall not affect the supplementary group list in any way.59311 **RETURN VALUE**59312           Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
59313 indicate the error.59314 **ERRORS**59315           The *setegid()* function shall fail if:59316           [EINVAL]           The value of the *gid* argument is invalid and is not supported by the  
59317 implementation.59318           [EPERM]           The process does not have appropriate privileges and *gid* does not match the  
59319 real group ID or the saved set-group-ID.59320 **EXAMPLES**

59321           None.

59322 **APPLICATION USAGE**

59323           None.

59324 **RATIONALE**59325           Refer to the RATIONALE section in *setuid()*.59326 **FUTURE DIRECTIONS**

59327           None.

59328 **SEE ALSO**59329           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

59330           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

59331 **CHANGE HISTORY**

59332           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

59333 **NAME**

59334            setenv — add or change environment variable

59335 **SYNOPSIS**

```
59336 CX       #include <stdlib.h>
59337       int setenv(const char *envname, const char *envval, int overwrite);
```

59338 **DESCRIPTION**

59339       The *setenv()* function shall update or add a variable in the environment of the calling process.  
 59340       The *envname* argument points to a string containing the name of an environment variable to be  
 59341       added or altered. The environment variable shall be set to the value to which *envval* points. The  
 59342       function shall fail if *envname* points to a string which contains an '=' character. If the  
 59343       environment variable named by *envname* already exists and the value of *overwrite* is non-zero,  
 59344       the function shall return success and the environment shall be updated. If the environment  
 59345       variable named by *envname* already exists and the value of *overwrite* is zero, the function shall  
 59346       return success and the environment shall remain unchanged.

59347       If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is  
 59348       undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

59349       The strings described by *envname* and *envval* are copied by this function.

59350       The *setenv()* function need not be thread-safe.

59351 **RETURN VALUE**

59352       Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
 59353       indicate the error, and the environment shall be unchanged.

59354 **ERRORS**

59355       The *setenv()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 59356 | [EINVAL] | The <i>name</i> argument is a null pointer, points to an empty string, or points to a string containing an '=' character. |
| 59357 |          |   |
| 59358 | [ENOMEM] | Insufficient memory was available to add a variable or its value to the environment.                                      |
| 59359 |          |   |

59360 **EXAMPLES**

59361       None.

59362 **APPLICATION USAGE**

59363       See *exec()* for restrictions on changing the environment in multi-threaded applications.

59364 **RATIONALE**

59365       Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if  
 59366       the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an  
 59367       obsolete copy of the environment (as may any other copy of *environ*). However, other than the  
 59368       aforementioned restriction, the standard developers intended that the traditional method of  
 59369       walking through the environment by way of the *environ* pointer must be supported.

59370       It was decided that *setenv()* should be required by this version because it addresses a piece of  
 59371       missing functionality, and does not impose a significant burden on the implementor.

59372       There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*  
 59373       function should be required as a mandatory function. The *setenv()* function was chosen because  
 59374       it permitted the implementation of the *unsetenv()* function to delete environmental variables,  
 59375       without specifying an additional interface. The *putenv()* function is available as part of the XSI  
 59376       option.

**setenv()**

59377 The standard developers considered requiring that *setenv()* indicate an error when a call to it  
59378 would result in exceeding {ARG\_MAX}. The requirement was rejected since the condition might  
59379 be temporary, with the application eventually reducing the environment size. The ultimate  
59380 success or failure depends on the size at the time of a call to *exec*, which returns an indication of  
59381 this error condition.

**59382 FUTURE DIRECTIONS**

59383 None.

**59384 SEE ALSO**

59385 *exec*, *getenv()*, *unsetenv()*

59386 XBD [<stdlib.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

**59387 CHANGE HISTORY**

59388 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

59389 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in  
59390 the APPLICATION USAGE and SEE ALSO sections.

**59391 Issue 7**

59392 Austin Group Interpretation 1003.1-2001 #156 is applied.

59393 **NAME**

59394        seteuid — set effective user ID

59395 **SYNOPSIS**

```
59396        #include <unistd.h>
59397        int seteuid(uid_t uid);
```

59398 **DESCRIPTION**

59399        If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate  
 59400        privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID  
 59401        and saved set-user-ID shall remain unchanged.

59402        The *seteuid()* function shall not affect the supplementary group list in any way.

59403 **RETURN VALUE**

59404        Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 59405        indicate the error.

59406 **ERRORS**

59407        The *seteuid()* function shall fail if:

59408        [EINVAL]        The value of the *uid* argument is invalid and is not supported by the  
 59409        implementation.

59410        [EPERM]        The process does not have appropriate privileges and *uid* does not match the  
 59411        real user ID or the saved set-user-ID.

59412 **EXAMPLES**

59413        None.

59414 **APPLICATION USAGE**

59415        None.

59416 **RATIONALE**

59417        Refer to the RATIONALE section in *setuid()*.

59418 **FUTURE DIRECTIONS**

59419        None.

59420 **SEE ALSO**

59421        *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

59422        XBD <sys/types.h>, <unistd.h>

59423 **CHANGE HISTORY**

59424        First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

59425        IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial  
 59426        correction to the [EPERM] error in the ERRORS section.

**setgid()**59427 **NAME**

59428 setgid — set-group-ID

59429 **SYNOPSIS**

59430 #include &lt;unistd.h&gt;

59431 int setgid(gid\_t gid);

59432 **DESCRIPTION**59433 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,  
59434 and the saved set-group-ID of the calling process to *gid*.59435 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the  
59436 saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved  
59437 set-group-ID shall remain unchanged.59438 The *setgid()* function shall not affect the supplementary group list in any way.

59439 Any supplementary group IDs of the calling process shall remain unchanged.

59440 **RETURN VALUE**59441 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to  
59442 indicate the error.59443 **ERRORS**59444 The *setgid()* function shall fail if:59445 [EINVAL] The value of the *gid* argument is invalid and is not supported by the  
59446 implementation.59447 [EPERM] The process does not have appropriate privileges and *gid* does not match the  
59448 real group ID or the saved set-group-ID.59449 **EXAMPLES**

59450 None.

59451 **APPLICATION USAGE**

59452 None.

59453 **RATIONALE**59454 Refer to the RATIONALE section in *setuid()*.59455 **FUTURE DIRECTIONS**

59456 None.

59457 **SEE ALSO**59458 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*

59459 XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

59460 **CHANGE HISTORY**

59461 First released in Issue 1. Derived from Issue 1 of the SVID.

59462 **Issue 6**

59463 In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

59464 The following new requirements on POSIX implementations derive from alignment with the  
59465 Single UNIX Specification:

- 59466
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
59467 required for conforming implementations of previous POSIX specifications, it was not  
59468 required for UNIX applications.

- 59469 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS  
59470 requirement.

59471 The following changes were made to align with the IEEE P1003.1a draft standard:

- 59472 • The effects of `setgid()` in processes without appropriate privileges are changed.  
59473 • A requirement that the supplementary group list is not affected is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setgrent()**59474 **NAME**

59475 setgrent — reset the group database to the first entry

59476 **SYNOPSIS**

```
59477 xSI #include <grp.h>  
59478 void setgrent(void);
```

59479 **DESCRIPTION**59480 Refer to *endgrent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***sethostent()**

59481 **NAME**  
59482       sethostent — network host database functions

59483 **SYNOPSIS**  
59484       #include <netdb.h>  
59485       void sethostent(int *stayopen*);

59486 **DESCRIPTION**  
59487       Refer to *endhostent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setitimer()***System Interfaces*59488 **NAME**

59489 setitimer — set the value of an interval timer

59490 **SYNOPSIS**

59491 OB XSI #include &lt;sys/time.h&gt;

59492 int setitimer(int *which*, const struct itimerval \*restrict *value*,  
59493 struct itimerval \*restrict *ovalue*);59494 **DESCRIPTION**59495 Refer to *getitimer()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59496 **NAME**

59497 setjmp — set jump point for a non-local goto

59498 **SYNOPSIS**

```
59499 #include <setjmp.h>
59500 int setjmp(jmp_buf env);
```

59501 **DESCRIPTION**

59502 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 59503 conflict between the requirements described here and the ISO C standard is unintentional. This  
 59504 volume of POSIX.1-2008 defers to the ISO C standard.

59505 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by  
 59506 *longjmp()*.

59507 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in  
 59508 order to access an actual function, or a program defines an external identifier with the name  
 59509 *setjmp*, the behavior is undefined.

59510 An application shall ensure that an invocation of *setjmp()* appears in one of the following  
 59511 contexts only:

- 59512 • The entire controlling expression of a selection or iteration statement
- 59513 • One operand of a relational or equality operator with the other operand an integral  
 59514 constant expression, with the resulting expression being the entire controlling expression  
 59515 of a selection or iteration statement
- 59516 • The operand of a unary '!' operator with the resulting expression being the entire  
 59517 controlling expression of a selection or iteration
- 59518 • The entire expression of an expression statement (possibly cast to **void**)

59519 If the invocation appears in any other context, the behavior is undefined.

59520 **RETURN VALUE**

59521 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to  
 59522 *longjmp()*, *setjmp()* shall return a non-zero value.

59523 **ERRORS**

59524 No errors are defined.

59525 **EXAMPLES**

59526 None.

59527 **APPLICATION USAGE**

59528 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-  
 59529 level subroutine of a program.

59530 **RATIONALE**

59531 None.

59532 **FUTURE DIRECTIONS**

59533 None.

59534 **SEE ALSO**

59535 *longjmp()*, *sigsetjmp()*

59536 XBD <setjmp.h>

## setjmp()

### 59537 **CHANGE HISTORY**

59538 First released in Issue 1. Derived from Issue 1 of the SVID.

### 59539 **Issue 6**

59540 The normative text is updated to avoid use of the term “must” for application requirements.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59541 **NAME**59542 setkey — set encoding key (**CRYPT**)59543 **SYNOPSIS**

```
59544 XSI #include <stdlib.h>
59545 void setkey(const char *key);
```

59546 **DESCRIPTION**

59547 The *setkey()* function provides access to an implementation-defined encoding algorithm. The  
 59548 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical  
 59549 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is  
 59550 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used  
 59551 with the algorithm to encode a string *block* passed to *encrypt()*.

59552 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to  
 59553 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on  
 59554 return, an error has occurred.

59555 The *setkey()* function need not be thread-safe.

59556 **RETURN VALUE**

59557 No values are returned.

59558 **ERRORS**59559 The *setkey()* function shall fail if:

59560 [ENOSYS] The functionality is not supported on this implementation.

59561 **EXAMPLES**

59562 None.

59563 **APPLICATION USAGE**

59564 Decoding need not be implemented in all environments. This is related to government  
 59565 restrictions in some countries on encryption and decryption routines. Historical practice has  
 59566 been to ship a different version of the encryption library without the decryption feature in the  
 59567 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

59568 **RATIONALE**

59569 None.

59570 **FUTURE DIRECTIONS**

59571 None.

59572 **SEE ALSO**59573 *crypt()*, *encrypt()*

59574 XBD &lt;stdlib.h&gt;

59575 **CHANGE HISTORY**

59576 First released in Issue 1. Derived from Issue 1 of the SVID.

59577 **Issue 5**59578 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.59579 **Issue 7**

59580 Austin Group Interpretation 1003.1-2001 #156 is applied.

## setlocale()

## 59581 NAME

59582 setlocale — set program locale

## 59583 SYNOPSIS

59584 #include &lt;locale.h&gt;

59585 char \*setlocale(int *category*, const char \**locale*);

## 59586 DESCRIPTION

59587 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 59588 conflict between the requirements described here and the ISO C standard is unintentional. This  
 59589 volume of POSIX.1-2008 defers to the ISO C standard.

59590 The *setlocale()* function selects the appropriate piece of the locale of the process, as specified by  
 59591 the *category* and *locale* arguments, and may be used to change or query the entire locale of the  
 59592 process or portions thereof. The value *LC\_ALL* for *category* names the entire locale of the process;  
 59593 other values for *category* name only a part of the locale of the process:

59594 *LC\_COLLATE* Affects the behavior of regular expressions and the collation functions.

59595 *LC\_CTYPE* Affects the behavior of regular expressions, character classification, character  
 59596 conversion functions, and wide-character functions.

59597 CX *LC\_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or  
 59598 negative responses.

59599 XSI It also affects what strings are given by commands and utilities as affirmative  
 59600 or negative responses, and the content of messages.

59601 *LC\_MONETARY* Affects the behavior of functions that handle monetary values.

59602 *LC\_NUMERIC* Affects the behavior of functions that handle numeric values.

59603 *LC\_TIME* Affects the behavior of the time conversion functions.

59604 The *locale* argument is a pointer to a character string containing the required setting of *category*.  
 59605 The contents of this string are implementation-defined. In addition, the following preset values  
 59606 of *locale* are defined for all settings of *category*:

59607 CX "POSIX" Specifies the minimal environment for C-language translation called the  
 59608 POSIX locale. If *setlocale()* is not invoked, the POSIX locale is the default at  
 59609 entry to *main()*.

59610 "C" Equivalent to "POSIX".

59611 CX "" Specifies an implementation-defined native environment. The determination  
 59612 of the name of the new locale for the specified category depends on the value  
 59613 of the associated environment variables, *LC\_\** and *LANG*; see XBD Chapter 7  
 59614 (on page 135) and Chapter 8 (on page 173).

59615 A null pointer Used to direct *setlocale()* to query the current internationalized environment  
 59616 and return the name of the locale.

59617 CX Setting all of the categories of the locale of the process is similar to successively setting each  
 59618 individual category of the locale of the process, except that all error checking is done before any  
 59619 actions are performed. To set all the categories of the locale of the process, *setlocale()* is invoked  
 59620 as:

```
59621 setlocale(LC_ALL, "");
```

59622 In this case, *setlocale()* shall first verify that the values of all the environment variables it needs  
 59623 according to the precedence rules (described in XBD Chapter 8, on page 173) indicate supported

59624 locales. If the value of any of these environment variable searches yields a locale that is not  
 59625 supported (and non-null), *setlocale()* shall return a null pointer and the locale of the process shall  
 59626 not be changed. If all environment variables name supported locales, *setlocale()* shall proceed as  
 59627 if it had been called for each category, using the appropriate value from the associated  
 59628 environment variable or from the implementation-defined default if there is no such value.

59629 The locale state is common to all threads within a process.

#### 59630 RETURN VALUE

59631 Upon successful completion, *setlocale()* shall return the string associated with the specified  
 59632 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the locale of the  
 59633 process is not changed.

59634 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the  
 59635 category for the current locale of the process. The locale of the process shall not be changed.

59636 The string returned by *setlocale()* is such that a subsequent call with that string and its associated  
 59637 category shall restore that part of the locale of the process. The application shall not modify the  
 59638 string returned which may be overwritten by a subsequent call to *setlocale()*.

#### 59639 ERRORS

59640 No errors are defined.

#### 59641 EXAMPLES

59642 None.

#### 59643 APPLICATION USAGE

59644 The following code illustrates how a program can initialize the international environment for  
 59645 one language, while selectively modifying the locale of the process such that regular expressions  
 59646 and string operations can be applied to text recorded in a different language:

```
59647 setlocale(LC_ALL, "De");
59648 setlocale(LC_COLLATE, "FrédicT");
```

59649 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can  
 59650 be done by calling *setlocale()* as follows:

```
59651 setlocale(LC_ALL, "");
```

59652 Changing the setting of *LC\_MESSAGES* has no effect on catalogs that have already been opened  
 59653 by calls to *catopen()*.

#### 59654 RATIONALE

59655 The ISO C standard defines a collection of functions to support internationalization. One of the  
 59656 most significant aspects of these functions is a facility to set and query the *international*  
 59657 *environment*. The international environment is a repository of information that affects the  
 59658 behavior of certain functionality, namely:

- 59659 1. Character handling
- 59660 2. Collating
- 59661 3. Date/time formatting
- 59662 4. Numeric editing
- 59663 5. Monetary formatting
- 59664 6. Messaging

59665 The *setlocale()* function provides the application developer with the ability to set all or portions,

**setlocale()**

59666 called *categories*, of the international environment. These categories correspond to the areas of  
 59667 functionality mentioned above. The syntax for *setlocale()* is as follows:

```
59668 char *setlocale(int category, const char *locale);
```

59669 where *category* is the name of one of following categories, namely:

```
59670     LC_COLLATE
59671     LC_CTYPE
59672     LC_MESSAGES
59673     LC_MONETARY
59674     LC_NUMERIC
59675     LC_TIME
```

59676 In addition, a special value called *LC\_ALL* directs *setlocale()* to set all categories.

59677 There are two primary uses of *setlocale()*:

- 59678 1. Querying the international environment to find out what it is set to
- 59679 2. Setting the international environment, or *locale*, to a specific value

59680 The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe  
 59681 the behavior in words, examples are used to illustrate the behavior of specific uses.

59682 To query the international environment, *setlocale()* is invoked with a specific category and the  
 59683 null pointer as the locale. The null pointer is a special directive to *setlocale()* that tells it to query  
 59684 rather than set the international environment. The following syntax is used to query the name of  
 59685 the international environment:

```
59686 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
59687          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

59688 The *setlocale()* function shall return the string corresponding to the current international  
 59689 environment. This value may be used by a subsequent call to *setlocale()* to reset the international  
 59690 environment to this value. However, it should be noted that the return value from *setlocale()*  
 59691 may be a pointer to a static area within the function and is not guaranteed to remain unchanged  
 59692 (that is, it may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of  
 59693 calling *setlocale()* is to save the value of the current international environment so it can be  
 59694 changed and reset later, the return value should be copied to an array of **char** in the calling  
 59695 program.

59696 There are three ways to set the international environment with *setlocale()*:

59697 *setlocale(category, string)*

59698 This usage sets a specific *category* in the international environment to a specific value  
 59699 corresponding to the value of the *string*. A specific example is provided below:

```
59700 setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

59701 In this example, all categories of the international environment are set to the locale  
 59702 corresponding to the string "fr\_FR.ISO-8859-1", or to the French language as spoken in  
 59703 France using the ISO/IEC 8859-1:1998 standard codeset.

59704 If the string does not correspond to a valid locale, *setlocale()* shall return a null pointer and  
 59705 the international environment is not changed. Otherwise, *setlocale()* shall return the name of  
 59706 the locale just set.

- 59707 *setlocale(category, "C")*  
 59708 The ISO C standard states that one locale must exist on all conforming implementations.  
 59709 The name of the locale is C and corresponds to a minimal international environment needed  
 59710 to support the C programming language.
- 59711 *setlocale(category, "")*  
 59712 This sets a specific category to an implementation-defined default. This corresponds to the  
 59713 value of the environment variables.
- 59714 **FUTURE DIRECTIONS**  
 59715 None.
- 59716 **SEE ALSO**  
 59717 *exec, fprintf(), fscanf(), isalnum(), isalpha(), isblank(), iscntrl(), isdigit(), isgraph(), islower(),*  
 59718 *isprint(), ispunct(), isspace(), isupper(), iswalnum(), iswalphabet(), iswblank(), iswcntrl(), iswctype(),*  
 59719 *iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(),*  
 59720 *isxdigit(), localeconv(), mblen(), mbstowcs(), mbtowc(), nl\_langinfo(), setlocale(), strcoll(),*  
 59721 *strerror(), strfmon(), strsignal(), strtod(), strxfrm(), tolower(), toupper(), towlower(), towupper(),*  
 59722 *uselocale(), wscoll(), wcstod(), wcstombs(), wcsxfrm(), wctomb()*
- 59723 XBD Chapter 7 (on page 135), Chapter 8 (on page 173), [<langinfo.h>](#), [<locale.h>](#)
- 59724 **CHANGE HISTORY**  
 59725 First released in Issue 3.
- 59726 **Issue 5**  
 59727 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 59728 **Issue 6**  
 59729 Extensions beyond the ISO C standard are marked.  
 59730 The normative text is updated to avoid use of the term “must” for application requirements.  
 59731 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the  
 59732 DESCRIPTION to clarify the behavior of:  
 59733 `setlocale(LC_ALL, "");`
- 59734 **Issue 7**  
 59735 Functionality relating to the Threads option is moved to the Base.

**setlogmask()***System Interfaces*59736 **NAME**

59737 setlogmask — set the log priority mask

59738 **SYNOPSIS**

```
59739 xSI #include <syslog.h>  
59740 int setlogmask(int maskpri);
```

59741 **DESCRIPTION**59742 Refer to *closelog()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***setnetent()**59743 **NAME**

59744 setnetent — network database function

59745 **SYNOPSIS**

59746 #include &lt;netdb.h&gt;

59747 void setnetent(int stayopen);

59748 **DESCRIPTION**59749 Refer to *endnetent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setpgid()**59750 **NAME**

59751 setpgid — set process group ID for job control

59752 **SYNOPSIS**

59753 #include &lt;unistd.h&gt;

59754 int setpgid(pid\_t pid, pid\_t pgid);

59755 **DESCRIPTION**59756 The *setpgid()* function shall either join an existing process group or create a new process group  
59757 within the session of the calling process.

59758 The process group ID of a session leader shall not change.

59759 Upon successful completion, the process group ID of the process with a process ID that matches  
59760 *pid* shall be set to *pgid*.59761 As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0,  
59762 the process ID of the indicated process shall be used.59763 **RETURN VALUE**59764 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*  
59765 shall be set to indicate the error.59766 **ERRORS**59767 The *setpgid()* function shall fail if:59768 [EACCES] The value of the *pid* argument matches the process ID of a child process of the  
59769 calling process and the child process has successfully executed one of the *exec*  
59770 functions.59771 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by  
59772 the implementation.59773 [EPERM] The process indicated by the *pid* argument is a session leader.59774 [EPERM] The value of the *pid* argument matches the process ID of a child process of the  
59775 calling process and the child process is not in the same session as the calling  
59776 process.59777 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of  
59778 the process indicated by the *pid* argument and there is no process with a  
59779 process group ID that matches the value of the *pgid* argument in the same  
59780 session as the calling process.59781 [ESRCH] The value of the *pid* argument does not match the process ID of the calling  
59782 process or of a child process of the calling process.59783 **EXAMPLES**

59784 None.

59785 **APPLICATION USAGE**

59786 None.

59787 **RATIONALE**59788 The *setpgid()* function shall group processes together for the purpose of signaling, placement in  
59789 foreground or background, and other job control actions.59790 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed  
59791 the specified new process group to assume any value. This presents certain security problems  
59792 and is more flexible than necessary to support job control.

59793 To provide tighter security, *setpgid()* only allows the calling process to join a process group  
59794 already in use inside its session or create a new process group whose process group ID was  
59795 equal to its process ID.

59796 When a job control shell spawns a new job, the processes in the job must be placed into a new  
59797 process group via *setpgid()*. There are two timing constraints involved in this action:

59798 1. The new process must be placed in the new process group before the appropriate  
59799 program is launched via one of the *exec* functions.

59800 2. The new process must be placed in the new process group before the shell can correctly  
59801 send signals to the new process group.

59802 To address these constraints, the following actions are performed. The new processes call  
59803 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first  
59804 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of  
59805 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that  
59806 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization  
59807 property was considered, but it was decided instead to merely allow the parent shell process to  
59808 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now  
59809 satisfied by having both the parent shell and the child attempt to adjust the process group of the  
59810 child process; it does not matter which succeeds first.

59811 Since it would be confusing to an application to have its process group change after it began  
59812 executing (that is, after *exec*), and because the child process would already have adjusted its  
59813 process group before this, the [EACCES] error was added to disallow this.

59814 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original  
59815 process group (the one in effect when the job control shell was executed). A job control shell  
59816 does this before returning control back to its parent when it is terminating or suspending itself as  
59817 a way of restoring its job control "state" back to what its parent would expect. (Note that the  
59818 original process group of the job control shell typically matches the process group of its parent,  
59819 but this is not necessarily always the case.)

#### 59820 FUTURE DIRECTIONS

59821 None.

#### 59822 SEE ALSO

59823 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*

59824 XBD [<sys/types.h>](#), [<unistd.h>](#)

#### 59825 CHANGE HISTORY

59826 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 59827 Issue 6

59828 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

59829 The following new requirements on POSIX implementations derive from alignment with the  
59830 Single UNIX Specification:

59831 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
59832 required for conforming implementations of previous POSIX specifications, it was not  
59833 required for UNIX applications.

59834 • The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be  
59835 defined in this version. This is a FIPS requirement.

**setpgid()**

59836 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in  
59837 the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the  
59838 process ID of the indicated process shall be used”. This change reverts the wording to as in the  
59839 ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59840 **NAME**

59841 setpgrp — set the process group ID

59842 **SYNOPSIS**

```
59843 OB XSI #include <unistd.h>
59844 pid_t setpgrp(void);
```

59845 **DESCRIPTION**

59846 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the  
 59847 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then the  
 59848 new session has no controlling terminal.

59849 The *setpgrp()* function has no effect when the calling process is a session leader.

59850 **RETURN VALUE**

59851 Upon completion, *setpgrp()* shall return the process group ID.

59852 **ERRORS**

59853 No errors are defined.

59854 **EXAMPLES**

59855 None.

59856 **APPLICATION USAGE**

59857 It is unspecified whether this function behaves as *setpgid(0,0)* or *setsid()* unless the process is  
 59858 already a session leader. Therefore, applications are encouraged to use *setpgid()* or *setsid()* as  
 59859 appropriate.

59860 **RATIONALE**

59861 None.

59862 **FUTURE DIRECTIONS**

59863 The *setpgrp()* function may be removed in a future version.

59864 **SEE ALSO**

59865 *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*

59866 XBD <unistd.h>

59867 **CHANGE HISTORY**

59868 First released in Issue 4, Version 2.

59869 **Issue 5**

59870 Moved from X/OPEN UNIX extension to BASE.

59871 **Issue 7**

59872 The *setpgrp()* function is marked obsolescent.

**setpriority()**

System Interfaces

59873 **NAME**

59874           setpriority — set the nice value

59875 **SYNOPSIS**

```
59876 XSI       #include <sys/resource.h>  
59877           int setpriority(int which, id_t who, int nice);
```

59878 **DESCRIPTION**59879           Refer to *getpriority()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***setprotoent()**

59880 **NAME**  
59881       setprotoent — network protocol database functions

59882 **SYNOPSIS**  
59883       #include <netdb.h>  
59884       void setprotoent(int *stayopen*);

59885 **DESCRIPTION**  
59886       Refer to *endprotoent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setpwent()**59887 **NAME**

59888           setpwent — user database function

59889 **SYNOPSIS**

59890 XSI       #include &lt;pwd.h&gt;

59891       void setpwent(void);

59892 **DESCRIPTION**59893       Refer to *endpwent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59894 **NAME**

59895 setregid — set real and effective group IDs

59896 **SYNOPSIS**

```
59897 XSI #include <unistd.h>
59898 int setregid(gid_t rgid, gid_t egid);
```

59899 **DESCRIPTION**59900 The *setregid()* function shall set the real and effective group IDs of the calling process.59901 If *rgid* is  $-1$ , the real group ID shall not be changed; if *egid* is  $-1$ , the effective group ID shall not  
59902 be changed.

59903 The real and effective group IDs may be set to different values in the same call.

59904 Only a process with appropriate privileges can set the real group ID and the effective group ID  
59905 to any valid value.59906 A non-privileged process can set either the real group ID to the saved set-group-ID from one of  
59907 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group  
59908 ID.59909 If the real group ID is being set (*rgid* is not  $-1$ ), or the effective group ID is being set to a value  
59910 not equal to the real group ID, then the saved set-group-ID of the current process shall be set  
59911 equal to the new effective group ID.

59912 Any supplementary group IDs of the calling process remain unchanged.

59913 **RETURN VALUE**59914 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
59915 indicate the error, and neither of the group IDs are changed.59916 **ERRORS**59917 The *setregid()* function shall fail if:59918 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.59919 [EPERM] The process does not have appropriate privileges and a change other than  
59920 changing the real group ID to the saved set-group-ID, or changing the  
59921 effective group ID to the real group ID or the saved set-group-ID, was  
59922 requested.59923 **EXAMPLES**

59924 None.

59925 **APPLICATION USAGE**59926 If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can  
59927 only set its effective group ID back to the previous value if *rgid* was  $-1$  in the *setregid()* call, since  
59928 the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the  
59929 *setregid()* call, then the saved set-group-ID will also have been changed to the real user ID.59930 **RATIONALE**59931 Earlier versions of this standard did not specify whether the saved set-group-ID was affected by  
59932 *setregid()* calls. This version specifies common existing practice that constitutes an important  
59933 security feature. The ability to set both the effective group ID and saved set-group-ID to be the  
59934 same as the real group ID means that any security weakness in code that is executed after that  
59935 point cannot result in malicious code being executed with the previous effective group ID.  
59936 Privileged applications could already do this using just *setgid()*, but for non-privileged

**setregid()**

59937 applications the only standard method available is to use this feature of *setregid()*.

**59938 FUTURE DIRECTIONS**

59939 None.

**59940 SEE ALSO**

59941 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*

59942 XBD <[unistd.h](#)>

**59943 CHANGE HISTORY**

59944 First released in Issue 4, Version 2.

**59945 Issue 5**

59946 Moved from X/OPEN UNIX extension to BASE.

59947 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the *exec* family of functions, not just *execve()*.

**59949 Issue 7**

59950 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID.

59951

59952 **NAME**

59953 setreuid — set real and effective user IDs

59954 **SYNOPSIS**

```
59955 XSI #include <unistd.h>
59956 int setreuid(uid_t ruid, uid_t euid);
```

59957 **DESCRIPTION**

59958 The *setreuid()* function shall set the real and effective user IDs of the current process to the  
 59959 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is  $-1$ , the corresponding effective  
 59960 or real user ID of the current process shall be left unchanged.

59961 A process with appropriate privileges can set either ID to any value. An unprivileged process  
 59962 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or  
 59963 saved user ID of the process.

59964 If the real user ID is being set (*ruid* is not  $-1$ ), or the effective user ID is being set to a value not  
 59965 equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to  
 59966 the new effective user ID.

59967 It is unspecified whether a process without appropriate privileges is permitted to change the real  
 59968 user ID to match the current effective user ID or saved set-user-ID of the process.

59969 **RETURN VALUE**

59970 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 59971 indicate the error.

59972 **ERRORS**

59973 The *setreuid()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 59974 | [EINVAL] | The value of the <i>ruid</i> or <i>euid</i> argument is invalid or out-of-range.   |
| 59975 | [EPERM]  | The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation. |

59979 **EXAMPLES**59980 **Setting the Effective User ID to the Real User ID**

59981 The following example sets the effective user ID of the calling process to the real user ID, so that  
 59982 files created later will be owned by the current user. It also sets the saved set-user-ID to the real  
 59983 user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
59984 #include <unistd.h>
59985 #include <sys/types.h>
59986 ...
59987 setreuid(getuid(), getuid());
59988 ...
```

59989 **APPLICATION USAGE**

59990 None.

**setreuid()**59991 **RATIONALE**

59992 Earlier versions of this standard did not specify whether the saved set-user-ID was affected by  
59993 *setreuid()* calls. This version specifies common existing practice that constitutes an important  
59994 security feature. The ability to set both the effective user ID and saved set-user-ID to be the same  
59995 as the real user ID means that any security weakness in code that is executed after that point  
59996 cannot result in malicious code being executed with the previous effective user ID. Privileged  
59997 applications could already do this using just *setuid()*, but for non-privileged applications the  
59998 only standard method available is to use this feature of *setreuid()*.

59999 **FUTURE DIRECTIONS**

60000 None.

60001 **SEE ALSO**

60002 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*

60003 XBD <[unistd.h](#)>

60004 **CHANGE HISTORY**

60005 First released in Issue 4, Version 2.

60006 **Issue 5**

60007 Moved from X/OPEN UNIX extension to BASE.

60008 **Issue 7**

60009 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved  
60010 set-user-ID to be the same as the real user ID.

60011 **NAME**

60012 setrlimit — control maximum resource consumption

60013 **SYNOPSIS**

```
60014 XSI #include <sys/resource.h>  
60015 int setrlimit(int resource, const struct rlimit *rlp);
```

60016 **DESCRIPTION**60017 Refer to *getrlimit()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setservent()**

60018 **NAME**  
60019       setservent — network services database functions

60020 **SYNOPSIS**  
60021       #include <netdb.h>  
60022       void setservent(int *stayopen*);

60023 **DESCRIPTION**  
60024       Refer to *endservent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

60025 **NAME**

60026 setsid — create session and set process group ID

60027 **SYNOPSIS**

60028 #include &lt;unistd.h&gt;

60029 pid\_t setsid(void);

60030 **DESCRIPTION**

60031 The *setsid()* function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

60037 **RETURN VALUE**

60038 Upon successful completion, *setsid()* shall return the value of the new process group ID of the calling process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

60040 **ERRORS**60041 The *setsid()* function shall fail if:

60042 [EPERM] The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

60045 **EXAMPLES**

60046 None.

60047 **APPLICATION USAGE**

60048 None.

60049 **RATIONALE**

60050 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp()*; only one process group may be part of a login session.

60053 Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2008 adds the concept of a session that is created via *setsid()*. The *setsid()* function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid()* function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function guarantees that the process ID of the new process does not match any existing process group ID.

60062 **FUTURE DIRECTIONS**

60063 None.

60064 **SEE ALSO**60065 *getsid()*, *setpgid()*, *setpgrp()*

60066 XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

**setsid()**60067 **CHANGE HISTORY**

60068 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60069 **Issue 6**

60070 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

60071 The following new requirements on POSIX implementations derive from alignment with the  
60072 Single UNIX Specification:

- 60073 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>`  
60074 was required for conforming implementations of previous POSIX specifications, it was not  
60075 required for UNIX applications.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

60076 **NAME**

60077 setsockopt — set the socket options

60078 **SYNOPSIS**

60079 #include &lt;sys/socket.h&gt;

```
60080 int setsockopt(int socket, int level, int option_name,
60081               const void *option_value, socklen_t option_len);
```

60082 **DESCRIPTION**

60083 The *setsockopt()* function shall set the option specified by the *option\_name* argument, at the  
 60084 protocol level specified by the *level* argument, to the value pointed to by the *option\_value*  
 60085 argument for the socket associated with the file descriptor specified by the *socket* argument.

60086 The *level* argument specifies the protocol level at which the option resides. To set options at the  
 60087 socket level, specify the *level* argument as SOL\_SOCKET. To set options at other levels, supply  
 60088 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate  
 60089 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO\_TCP  
 60090 as defined in the <netinet/in.h> header.

60091 The *option\_name* argument specifies a single option to set. It can be one of the socket-level  
 60092 options defined in <sys/socket.h> and described in Section 2.10.16 (on page 522). If *setsockopt()*  
 60093 is called with *option\_name* equal to SO\_ACCEPTCONN, SO\_ERROR, or SO\_TYPE, the behavior  
 60094 is unspecified.

60095 **RETURN VALUE**

60096 Upon successful completion, *setsockopt()* shall return 0. Otherwise, -1 shall be returned and  
 60097 *errno* set to indicate the error.

60098 **ERRORS**

60099 The *setsockopt()* function shall fail if:

60100 [EBADF] The *socket* argument is not a valid file descriptor.

60101 [EDOM] The send and receive timeout values are too big to fit into the timeout fields in  
 60102 the socket structure.

60103 [EINVAL] The specified option is invalid at the specified socket level or the socket has  
 60104 been shut down.

60105 [EISCONN] The socket is already connected, and a specified option cannot be set while the  
 60106 socket is connected.

60107 [ENOPROTOPT]

60108 The option is not supported by the protocol.

60109 [ENOTSOCK] The *socket* argument does not refer to a socket.

60110 The *setsockopt()* function may fail if:

60111 [ENOMEM] There was insufficient memory available for the operation to complete.

60112 [ENOBUFS] Insufficient resources are available in the system to complete the call.

**setsockopt()**60113 **EXAMPLES**

60114 None.

60115 **APPLICATION USAGE**

60116 The *setsockopt()* function provides an application program with the means to control socket  
60117 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,  
60118 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options  
60119 available to *setsockopt()*.

60120 Options may exist at multiple protocol levels. The SO\_ options are always present at the  
60121 uppermost socket level.

60122 **RATIONALE**

60123 None.

60124 **FUTURE DIRECTIONS**

60125 None.

60126 **SEE ALSO**60127 [Section 2.10](#) (on page 517), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*60128 XBD `<netinet/in.h>`, `<sys/socket.h>`60129 **CHANGE HISTORY**

60130 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

60131 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the SO\_LINGER  
60132 option in the DESCRIPTION to refer to the calling thread rather than the process.

60133 **Issue 7**

60134 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options  
60135 that is now in [Section 2.10.16](#) (on page 522).

*System Interfaces***setstate()**60136 **NAME**60137        **setstate** — switch pseudo-random number generator state arrays60138 **SYNOPSIS**

```
60139 XSI        #include <stdlib.h>  
60140        char *setstate(char *state);
```

60141 **DESCRIPTION**60142        Refer to *initstate()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setuid()**60143 **NAME**

60144        setuid — set user ID

60145 **SYNOPSIS**

```
60146        #include <unistd.h>
60147        int setuid(uid_t uid);
```

60148 **DESCRIPTION**

60149        If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and  
 60150        the saved set-user-ID of the calling process to *uid*.

60151        If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the  
 60152        saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-  
 60153        user-ID shall remain unchanged.

60154        The *setuid()* function shall not affect the supplementary group list in any way.

60155 **RETURN VALUE**

60156        Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 60157        indicate the error.

60158 **ERRORS**

60159        The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more  
 60160        of the following are true:

60161        [EINVAL]        The value of the *uid* argument is invalid and not supported by the  
 60162        implementation.

60163        [EPERM]        The process does not have appropriate privileges and *uid* does not match the  
 60164        real user ID or the saved set-user-ID.

60165 **EXAMPLES**

60166        None.

60167 **APPLICATION USAGE**

60168        None.

60169 **RATIONALE**

60170        The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged  
 60171        processes reflect the behavior of different historical implementations. For portability, it is  
 60172        recommended that new non-privileged applications use the *setuid()* and *setgid()* functions  
 60173        instead.

60174        The saved set-user-ID capability allows a program to regain the effective user ID established at  
 60175        the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the  
 60176        effective group ID established at the last *exec* call. These capabilities are derived from System V.  
 60177        Without them, a program might have to run as superuser in order to perform the same  
 60178        functions, because superuser can write on the user's files. This is a problem because such a  
 60179        program can write on any user's files, and so must be carefully written to emulate the  
 60180        permissions of the calling process properly. In System V, these capabilities have traditionally  
 60181        been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The  
 60182        fact that the behavior of those functions was different for privileged processes made them  
 60183        difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently  
 60184        for privileged and unprivileged users. When the caller had appropriate privileges, the function  
 60185        set the real user ID, effective user ID, and saved set-user ID of the calling process on  
 60186        implementations that supported it. When the caller did not have appropriate privileges, the  
 60187        function set only the effective user ID, subject to permission checks. The former use is generally  
 60188        needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

60189 scope of POSIX.1-2008. These utilities wish to change the user ID irrevocably to a new value,  
60190 generally that of an unprivileged user. The latter use is needed for conforming applications that  
60191 are installed with the set-user-ID bit and need to perform operations using the real user ID.

60192 POSIX.1-2008 augments the latter functionality with a mandatory feature named  
60193 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user  
60194 ID back and forth between the values of its *exec*-time real user ID and effective user ID.  
60195 Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this  
60196 feature to work properly when it happened to be executed with (implementation-defined)  
60197 appropriate privileges. Furthermore, the application did not even have a means to tell whether it  
60198 had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as  
60199 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2008.  
60200 However, there are implementors who have been reluctant to support it given the limitation  
60201 described above.

60202 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which  
60203 always sets both the real and effective user IDs, like *setuid()* in POSIX.1-2008 for privileged  
60204 users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in POSIX.1-2008  
60205 for non-privileged users). This separation of functionality into distinct functions seems desirable.  
60206 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of  
60207 switching the effective user ID back and forth via *setreuid()*, which permits reversing the real  
60208 and effective user IDs. This model seems less desirable than the saved set-user-ID because the  
60209 real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses  
60210 them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be  
60211 deprecated or removed.

60212 The solution here is:

- 60213 • Require that all implementations support the functionality of the saved set-user-ID, which  
60214 is set by the *exec* functions and by privileged calls to *setuid()*.
- 60215 • Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for  
60216 non-privileged and privileged processes.

60217 Historical systems have provided two mechanisms for a set-user-ID process to change its  
60218 effective user ID to be the same as its real user ID in such a way that it could return to the  
60219 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID,  
60220 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs.  
60221 The changes included in POSIX.1-2008 provide a new mechanism using *seteuid()* in conjunction  
60222 with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have  
60223 a saved set-user-ID for each process, and most of the behavior controlled by  
60224 `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined.  
60225 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally  
60226 be required to maintain compatibility with the older mechanisms previously supported by their  
60227 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS`  
60228 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID  
60229 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the  
60230 saved set-user-ID unmodified, the process would then have an effective user ID equal to the  
60231 original real user ID, and both real and saved set-user-ID would be equal to the original effective  
60232 user ID. In that state, the real user would be unable to kill the process, even though the effective  
60233 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS`  
60234 was used. This is obviously not acceptable. The alternative choice, which is used in at least one  
60235 implementation, is to change the saved set-user-ID to the effective user ID during most calls to  
60236 *setreuid()*. The standard developers considered that alternative to be less correct than the  
60237 retention of the old behavior of *kill()* in such systems. Current conforming applications shall

**setuid()**

60238 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to  
 60239 check the saved set-user-ID rather than the effective user ID.

60240 **FUTURE DIRECTIONS**

60241 None.

60242 **SEE ALSO**

60243 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*

60244 XBD [<sys/types.h>](#), [<unistd.h>](#)

60245 **CHANGE HISTORY**

60246 First released in Issue 1. Derived from Issue 1 of the SVID.

60247 **Issue 6**

60248 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

60249 The following new requirements on POSIX implementations derive from alignment with the  
 60250 Single UNIX Specification:

- 60251 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 60252 required for conforming implementations of previous POSIX specifications, it was not  
 60253 required for UNIX applications.
- 60254 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS  
 60255 requirement.

60256 The following changes were made to align with the IEEE P1003.1a draft standard:

- 60257 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 60258 • A requirement that the supplementary group list is not affected is added.

*System Interfaces***setutxent()**60259 **NAME**

60260 setutxent — reset the user accounting database to the first entry

60261 **SYNOPSIS**

```
60262 XSI #include <utmpx.h>  
60263 void setutxent(void);
```

60264 **DESCRIPTION**60265 Refer to *endutxent()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**setvbuf()**60266 **NAME**60267 `setvbuf` — assign buffering to a stream60268 **SYNOPSIS**60269 `#include <stdio.h>`60270 `int setvbuf(FILE *restrict stream, char *restrict buf, int type,`  
60271 `size_t size);`60272 **DESCRIPTION**60273 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
60274 conflict between the requirements described here and the ISO C standard is unintentional. This  
60275 volume of POSIX.1-2008 defers to the ISO C standard.60276 The `setvbuf()` function may be used after the stream pointed to by `stream` is associated with an  
60277 open file but before any other operation (other than an unsuccessful call to `setvbuf()`) is  
60278 performed on the stream. The argument `type` determines how `stream` shall be buffered, as  
60279 follows:

- 60280
- `{_IOFBF}` shall cause input/output to be fully buffered.
  - 60281 • `{_IOLBF}` shall cause input/output to be line buffered.
  - 60282 • `{_IONBF}` shall cause input/output to be unbuffered.

60283 If `buf` is not a null pointer, the array it points to may be used instead of a buffer allocated by  
60284 `setvbuf()` and the argument `size` specifies the size of the array; otherwise, `size` may determine the  
60285 size of a buffer allocated by the `setvbuf()` function. The contents of the array at any time are  
60286 unspecified.60287 For information about streams, see [Section 2.5](#) (on page 490).60288 **RETURN VALUE**60289 Upon successful completion, `setvbuf()` shall return 0. Otherwise, it shall return a non-zero value  
60290 **CX** if an invalid value is given for `type` or if the request cannot be honored, and may set `errno` to  
60291 indicate the error.60292 **ERRORS**60293 The `setvbuf()` function may fail if:60294 **CX** `[EBADF]` The file descriptor underlying `stream` is not valid.60295 **EXAMPLES**

60296 None.

60297 **APPLICATION USAGE**60298 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
60299 and then failing to close the stream in the same block.60300 With `setvbuf()`, allocating a buffer of `size` bytes does not necessarily imply that all of `size` bytes are  
60301 used for the buffer area.60302 Applications should note that many implementations only provide line buffering on input from  
60303 terminal devices.60304 **RATIONALE**

60305 None.

60306 **FUTURE DIRECTIONS**

60307 None.

60308 **SEE ALSO**60309 [Section 2.5](#) (on page 490), [fopen\(\)](#), [setbuf\(\)](#)60310 XBD [<stdio.h>](#)60311 **CHANGE HISTORY**

60312 First released in Issue 1. Derived from Issue 1 of the SVID.

60313 **Issue 6**

60314 Extensions beyond the ISO C standard are marked.

60315 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**shm\_open()**60316 **NAME**60317 shm\_open — open a shared memory object (**REALTIME**)60318 **SYNOPSIS**

```
60319 SHM #include <sys/mman.h>
60320 int shm_open(const char *name, int oflag, mode_t mode);
```

60321 **DESCRIPTION**

60322 The *shm\_open()* function shall establish a connection between a shared memory object and a file  
 60323 descriptor. It shall create an open file description that refers to the shared memory object and a  
 60324 file descriptor that refers to that open file description. The file descriptor is used by other  
 60325 functions to refer to that shared memory object. The *name* argument points to a string naming a  
 60326 shared memory object. It is unspecified whether the name appears in the file system and is  
 60327 visible to other functions that take pathnames as arguments. The *name* argument conforms to the  
 60328 construction rules for a pathname, except that the interpretation of <slash> characters other than  
 60329 the leading <slash> character in *name* is implementation-defined, and that the length limits for  
 60330 the *name* argument are implementation-defined and need not be the same as the pathname limits  
 60331 {PATH\_MAX} and {NAME\_MAX}. If *name* begins with the <slash> character, then processes  
 60332 calling *shm\_open()* with the same value of *name* refer to the same shared memory object, as long  
 60333 as that name has not been removed. If *name* does not begin with the <slash> character, the effect  
 60334 is implementation-defined.

60335 If successful, *shm\_open()* shall return a file descriptor for the shared memory object that is the  
 60336 lowest numbered file descriptor not currently open for that process. The open file description is  
 60337 new, and therefore the file descriptor does not share it with any other processes. It is unspecified  
 60338 whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated with the new file  
 60339 descriptor is set.

60340 The file status flags and file access modes of the open file description are according to the value  
 60341 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the  
 60342 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below  
 60343 in the value of *oflag*:

60344 O\_RDONLY Open for read access only.

60345 O\_RDWR Open for read or write access.

60346 Any combination of the remaining flags may be specified in the value of *oflag*:

60347 O\_CREAT If the shared memory object exists, this flag has no effect, except as noted  
 60348 under O\_EXCL below. Otherwise, the shared memory object is created. The  
 60349 user ID of the shared memory object shall be set to the effective user ID of the  
 60350 process. The group ID of the shared memory object shall be set to the effective  
 60351 group ID of the process; however, if the *name* argument is visible in the file  
 60352 system, the group ID may be set to the group ID of the containing directory.  
 60353 The permission bits of the shared memory object shall be set to the value of  
 60354 the *mode* argument except those set in the file mode creation mask of the  
 60355 process. When bits in *mode* other than the file permission bits are set, the effect  
 60356 is unspecified. The *mode* argument does not affect whether the shared memory  
 60357 object is opened for reading, for writing, or for both. The shared memory  
 60358 object has a size of zero.

60359 O\_EXCL If O\_EXCL and O\_CREAT are set, *shm\_open()* fails if the shared memory  
 60360 object exists. The check for the existence of the shared memory object and the  
 60361 creation of the object if it does not exist is atomic with respect to other

60362		processes executing <i>shm_open()</i> naming the same shared memory object with
60363		<code>O_EXCL</code> and <code>O_CREAT</code> set. If <code>O_EXCL</code> is set and <code>O_CREAT</code> is not set, the
60364		result is undefined.
60365	<code>O_TRUNC</code>	If the shared memory object exists, and it is successfully opened <code>O_RDWR</code> , the
60366		object shall be truncated to zero length and the mode and owner shall be
60367		unchanged by this function call. The result of using <code>O_TRUNC</code> with
60368		<code>O_RDONLY</code> is undefined.
60369		When a shared memory object is created, the state of the shared memory object, including all
60370		data associated with the shared memory object, persists until the shared memory object is
60371		unlinked and all other references are gone. It is unspecified whether the name and shared
60372		memory object state remain valid after a system reboot.
60373	<b>RETURN VALUE</b>	
60374		Upon successful completion, the <i>shm_open()</i> function shall return a non-negative integer
60375		representing the lowest numbered unused file descriptor. Otherwise, it shall return <code>-1</code> and set
60376		<i>errno</i> to indicate the error.
60377	<b>ERRORS</b>	
60378		The <i>shm_open()</i> function shall fail if:
60379	<code>[EACCES]</code>	The shared memory object exists and the permissions specified by <i>oflag</i> are
60380		denied, or the shared memory object does not exist and permission to create
60381		the shared memory object is denied, or <code>O_TRUNC</code> is specified and write
60382		permission is denied.
60383	<code>[EEXIST]</code>	<code>O_CREAT</code> and <code>O_EXCL</code> are set and the named shared memory object already
60384		exists.
60385	<code>[EINTR]</code>	The <i>shm_open()</i> operation was interrupted by a signal.
60386	<code>[EINVAL]</code>	The <i>shm_open()</i> operation is not supported for the given name.
60387	<code>[EMFILE]</code>	All file descriptors available to the process are currently open.
60388	<code>[ENFILE]</code>	Too many shared memory objects are currently open in the system.
60389	<code>[ENOENT]</code>	<code>O_CREAT</code> is not set and the named shared memory object does not exist.
60390	<code>[ENOSPC]</code>	There is insufficient space for the creation of the new shared memory object.
60391		The <i>shm_open()</i> function may fail if:
60392	<code>[ENAMETOOLONG]</code>	
60393		The length of the <i>name</i> argument exceeds <code>{_POSIX_PATH_MAX}</code> on systems
60394	XSI	that do not support the XSI option or exceeds <code>{_XOPEN_PATH_MAX}</code> on XSI
60395		systems, or has a pathname component that is longer than
60396	XSI	<code>{_POSIX_NAME_MAX}</code> on systems that do not support the XSI option or
60397		longer than <code>{_XOPEN_NAME_MAX}</code> on XSI systems.

**shm\_open()**60398 **EXAMPLES**60399 **Creating and Mapping a Shared Memory Object**

60400 The following code segment demonstrates the use of *shm\_open()* to create a shared memory  
60401 object which is then sized using *ftruncate()* before being mapped into the process address space  
60402 using *mmap()*:

```
60403 #include <unistd.h>
60404 #include <sys/mman.h>
60405 ...
60406 #define MAX_LEN 10000
60407 struct region {          /* Defines "structure" of shared memory */
60408     int len;
60409     char buf[MAX_LEN];
60410 };
60411 struct region *rptr;
60412 int fd;
60413 /* Create shared memory object and set its size */
60414 fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
60415 if (fd == -1)
60416     /* Handle error */;
60417 if (ftruncate(fd, sizeof(struct region)) == -1)
60418     /* Handle error */;
60419 /* Map shared memory object */
60420 rptr = mmap(NULL, sizeof(struct region),
60421             PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
60422 if (rptr == MAP_FAILED)
60423     /* Handle error */;
60424 /* Now we can refer to mapped region using fields of rptr;
60425    for example, rptr->len */
60426 ...
```

60427 **APPLICATION USAGE**

60428 None.

60429 **RATIONALE**

60430 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a  
60431 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared  
60432 Memory Objects option is supported, the *shm\_open()* function shall obtain a descriptor to the  
60433 shared memory object to be mapped.

60434 There is ample precedent for having a file descriptor represent several types of objects. In the  
60435 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.  
60436 Many implementations simply have an operations vector, which is indexed by the file descriptor  
60437 type and does very different operations. Note that in some cases the file descriptor passed to  
60438 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned  
60439 by alternate functions, such as *pipe()*. The latter technique is used by *shm\_open()*.

60440 Note that such shared memory objects can actually be implemented as mapped files. In both  
60441 cases, the size can be set after the open using *ftruncate()*. The *shm\_open()* function itself does not

60442 create a shared object of a specified size because this would duplicate an extant function that set  
60443 the size of an object referenced by a file descriptor.

60444 On implementations where memory objects are implemented using the existing file system, the  
60445 *shm\_open()* function may be implemented using a macro that invokes *open()*, and the  
60446 *shm\_unlink()* function may be implemented using a macro that invokes *unlink()*.

60447 For implementations without a permanent file system, the definition of the name of the memory  
60448 objects is allowed not to survive a system reboot. Note that this allows systems with a  
60449 permanent file system to implement memory objects as data structures internal to the  
60450 implementation as well.

60451 On implementations that choose to implement memory objects using memory directly, a  
60452 *shm\_open()* followed by an *ftruncate()* and *close()* can be used to preallocate a shared memory  
60453 area and to set the size of that preallocation. This may be necessary for systems without virtual  
60454 memory hardware support in order to ensure that the memory is contiguous.

60455 The set of valid open flags to *shm\_open()* was restricted to O\_RDONLY, O\_RDWR, O\_CREAT,  
60456 and O\_TRUNC because these could be easily implemented on most memory mapping systems.  
60457 This volume of POSIX.1-2008 is silent on the results if the implementation cannot supply the  
60458 requested file access because of implementation-defined reasons, including hardware ones.

60459 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the  
60460 implementation cannot complete a request.

60461 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the  
60462 implementation cannot comply with a requested mode because it conflicts with another  
60463 requested mode. An example might be that an application desires to open a memory object two  
60464 times, mapping different areas with different access modes. If the implementation cannot map a  
60465 single area into a process space in two places, which would be required if different access modes  
60466 were required for the two areas, then the implementation may inform the application at the time  
60467 of the second open.

60468 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the  
60469 implementation cannot comply with a requested mode at all. An example would be that the  
60470 hardware of the implementation cannot support write-only shared memory areas.

60471 On all implementations, it may be desirable to restrict the location of the memory objects to  
60472 specific file systems for performance (such as a RAM disk) or implementation-defined reasons  
60473 (shared memory supported directly only on certain file systems). The *shm\_open()* function may  
60474 be used to enforce these restrictions. There are a number of methods available to the application  
60475 to determine an appropriate name of the file or the location of an appropriate directory. One way  
60476 is from the environment via *getenv()*. Another would be from a configuration file.

60477 This volume of POSIX.1-2008 specifies that memory objects have initial contents of zero when  
60478 created. This is consistent with current behavior for both files and newly allocated memory. For  
60479 those implementations that use physical memory, it would be possible that such  
60480 implementations could simply use available memory and give it to the process uninitialized.  
60481 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,  
60482 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security  
60483 reasons. Thus, initializing memory objects to zero is required.

#### 60484 FUTURE DIRECTIONS

60485 A future version might require the *shm\_open()* and *shm\_unlink()* functions to have semantics  
60486 similar to normal file system operations.

**shm\_open()**60487 **SEE ALSO**60488 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_unlink()*, *umask()*

60489 XBD &lt;fcntl.h&gt;, &lt;sys/mman.h&gt;

60490 **CHANGE HISTORY**

60491 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60492 **Issue 6**60493 The *shm\_open()* function is marked as part of the Shared Memory Objects option.60494 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
60495 implementation does not support the Shared Memory Objects option.60496 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the  
60497 EXAMPLES section.60498 **Issue 7**60499 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and  
60500 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

60501 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

60502 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
60503 user ID and group ID of the shared memory object.

60504 **NAME**60505 shm\_unlink — remove a shared memory object (**REALTIME**)60506 **SYNOPSIS**

```
60507 SHM #include <sys/mman.h>
60508 int shm_unlink(const char *name);
```

60509 **DESCRIPTION**

60510 The *shm\_unlink()* function shall remove the name of the shared memory object named by the  
60511 string pointed to by *name*.

60512 If one or more references to the shared memory object exist when the object is unlinked, the  
60513 name shall be removed before *shm\_unlink()* returns, but the removal of the memory object  
60514 contents shall be postponed until all open and map references to the shared memory object have  
60515 been removed.

60516 Even if the object continues to exist after the last *shm\_unlink()*, reuse of the name shall  
60517 subsequently cause *shm\_open()* to behave as if no shared memory object of this name exists (that  
60518 is, *shm\_open()* will fail if O\_CREAT is not set, or will create a new shared memory object if  
60519 O\_CREAT is set).

60520 **RETURN VALUE**

60521 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
60522 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object  
60523 shall not be changed by this function call.

60524 **ERRORS**

60525 The *shm\_unlink()* function shall fail if:

60526 [EACCES] Permission is denied to unlink the named shared memory object.

60527 [ENOENT] The named shared memory object does not exist.

60528 The *shm\_unlink()* function may fail if:

60529 [ENAMETOOLONG]

60530 The length of the *name* argument exceeds {\_POSIX\_PATH\_MAX} on systems  
60531 XSI that do not support the XSI option or exceeds {\_XOPEN\_PATH\_MAX} on XSI  
60532 systems, or has a pathname component that is longer than  
60533 XSI {\_POSIX\_NAME\_MAX} on systems that do not support the XSI option or  
60534 longer than {\_XOPEN\_NAME\_MAX} on XSI systems. A call to *shm\_unlink()*  
60535 with a *name* argument that contains the same shared memory object name as  
60536 was previously used in a successful *shm\_open()* call shall not give an  
60537 [ENAMETOOLONG] error.

60538 **EXAMPLES**

60539 None.

60540 **APPLICATION USAGE**

60541 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual  
60542 fashion. Names of memory objects that were allocated with *shm\_open()* are deleted with  
60543 *shm\_unlink()*. Note that the actual memory object is not destroyed until the last close and  
60544 unmap on it have occurred if it was already in use.

**shm\_unlink()**60545 **RATIONALE**

60546 None.

60547 **FUTURE DIRECTIONS**60548 A future version might require the *shm\_open()* and *shm\_unlink()* functions to have semantics  
60549 similar to normal file system operations.60550 **SEE ALSO**60551 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*60552 XBD <[sys/mman.h](#)>60553 **CHANGE HISTORY**

60554 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60555 **Issue 6**60556 The *shm\_unlink()* function is marked as part of the Shared Memory Objects option.60557 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm\_unlink()*  
60558 will not attach to the old shared memory object.60559 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
60560 implementation does not support the Shared Memory Objects option.60561 **Issue 7**60562 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a  
60563 “shall fail” to a “may fail” error.

60564 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

60565 **NAME**

60566 shmat — XSI shared memory attach operation

60567 **SYNOPSIS**

```
60568 XSI #include <sys/shm.h>
60569 void *shmat(int shmid, const void *shmaddr, int shmflg);
```

60570 **DESCRIPTION**

60571 The *shmat()* function operates on XSI shared memory (see XBD Section 3.340, on page 88). It is  
 60572 unspecified whether this function interoperates with the realtime interprocess communication  
 60573 facilities defined in Section 2.8 (on page 497).

60574 The *shmat()* function attaches the shared memory segment associated with the shared memory  
 60575 identifier specified by *shmid* to the address space of the calling process. The segment is attached  
 60576 at the address specified by one of the following criteria:

- 60577 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected  
 60578 by the system.
- 60579 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is non-zero, the segment is  
 60580 attached at the address given by (*shmaddr* - ((*uintptr\_t*)*shmaddr* %SHMLBA)). The character  
 60581 ' % ' is the C-language remainder operator.
- 60582 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is 0, the segment is attached at  
 60583 the address given by *shmaddr*.
- 60584 • The segment is attached for reading if (*shmflg* &SHM\_RDONLY) is non-zero and the  
 60585 calling process has read permission; otherwise, if it is 0 and the calling process has read  
 60586 and write permission, the segment is attached for reading and writing.

60587 **RETURN VALUE**

60588 Upon successful completion, *shmat()* shall increment the value of *shm\_nattch* in the data  
 60589 structure associated with the shared memory ID of the attached shared memory segment and  
 60590 return the segment's start address.

60591 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return -1, and *errno*  
 60592 shall be set to indicate the error.

60593 **ERRORS**

60594 The *shmat()* function shall fail if:

- |   |          |  |
|---|----------|--|
| 60595<br>60596                            | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 496).  |
| 60597<br>60598<br>60599<br>60600<br>60601 | [EINVAL] | The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of ( <i>shmaddr</i> - (( <i>uintptr_t</i> ) <i>shmaddr</i> %SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, ( <i>shmflg</i> &SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory. |
| 60602<br>60603                            | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.  |
| 60604<br>60605                            | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.   |

**shmat()**60606 **EXAMPLES**

60607 None.

60608 **APPLICATION USAGE**

60609 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
60610 Application developers who need to use IPC should design their applications so that modules  
60611 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the  
60612 alternative interfaces.

60613 **RATIONALE**

60614 None.

60615 **FUTURE DIRECTIONS**

60616 None.

60617 **SEE ALSO**

60618 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*,  
60619 *shmget()*, *shm\_open()*, *shm\_unlink()*

60620 XBD [Section 3.340](#) (on page 88), [<sys/shm.h>](#)60621 **CHANGE HISTORY**

60622 First released in Issue 2. Derived from Issue 2 of the SVID.

60623 **Issue 5**

60624 Moved from SHARED MEMORY to BASE.

60625 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
60626 DIRECTIONS to a new APPLICATION USAGE section.

60627 **Issue 6**

60628 The Open Group Corrigendum U021/13 is applied.

60629 **NAME**

60630 shmctl — XSI shared memory control operations

60631 **SYNOPSIS**

```
60632 XSI #include <sys/shm.h>
60633 int shmctl(int shmid, int cmd, struct shm_ds *buf);
```

60634 **DESCRIPTION**

60635 The *shmctl()* function operates on XSI shared memory (see XBD Section 3.340, on page 88). It is  
 60636 unspecified whether this function interoperates with the realtime interprocess communication  
 60637 facilities defined in Section 2.8 (on page 497).

60638 The *shmctl()* function provides a variety of shared memory control operations as specified by  
 60639 *cmd*. The following values for *cmd* are available:

60640 **IPC\_STAT** Place the current value of each member of the **shm\_ds** data structure  
 60641 associated with *shmid* into the structure pointed to by *buf*. The contents of the  
 60642 structure are defined in **<sys/shm.h>**.

60643 **IPC\_SET** Set the value of the following members of the **shm\_ds** data structure  
 60644 associated with *shmid* to the corresponding value found in the structure  
 60645 pointed to by *buf*:

60646 shm\_perm.uid  
 60647 shm\_perm.gid  
 60648 shm\_perm.mode Low-order nine bits.

60649 IPC\_SET can only be executed by a process that has an effective user ID equal  
 60650 to either that of a process with appropriate privileges or to the value of  
 60651 *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_ds** data structure associated with  
 60652 *shmid*.

60653 **IPC\_RMID** Remove the shared memory identifier specified by *shmid* from the system and  
 60654 destroy the shared memory segment and **shm\_ds** data structure associated  
 60655 with it. IPC\_RMID can only be executed by a process that has an effective user  
 60656 ID equal to either that of a process with appropriate privileges or to the value  
 60657 of *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_ds** data structure associated  
 60658 with *shmid*.

60659 **RETURN VALUE**

60660 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 60661 indicate the error.

60662 **ERRORS**

60663 The *shmctl()* function shall fail if:

60664 [EACCES] The argument *cmd* is equal to IPC\_STAT and the calling process does not have  
 60665 read permission; see Section 2.7 (on page 496).

60666 [EINVAL] The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*  
 60667 is not a valid command.

60668 [EPERM] The argument *cmd* is equal to IPC\_RMID or IPC\_SET and the effective user ID  
 60669 of the calling process is not equal to that of a process with appropriate  
 60670 privileges and it is not equal to the value of *shm\_perm.cuid* or *shm\_perm.uid* in  
 60671 the data structure associated with *shmid*.

**shmctl()**

60672 The *shmctl()* function may fail if:

60673 [EOVERFLOW] The *cmd* argument is `IPC_STAT` and the *gid* or *uid* value is too large to be  
60674 stored in the structure pointed to by the *buf* argument.

**EXAMPLES**

60675 None.  
60676

**APPLICATION USAGE**

60677 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
60678 Application developers who need to use IPC should design their applications so that modules  
60679 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the  
60680 alternative interfaces.  
60681

**RATIONALE**

60682 None.  
60683

**FUTURE DIRECTIONS**

60684 None.  
60685

**SEE ALSO**

60686 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [shmctl\(\)](#), [shmdt\(\)](#), [shmget\(\)](#), [shm\\_open\(\)](#),  
60687 [shm\\_unlink\(\)](#)  
60688

60689 XBD [Section 3.340](#) (on page 88), [<sys/shm.h>](#)

**CHANGE HISTORY**

60690 First released in Issue 2. Derived from Issue 2 of the SVID.  
60691

**Issue 5**

60692 Moved from SHARED MEMORY to BASE.  
60693

60694 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
60695 DIRECTIONS to a new APPLICATION USAGE section.

60696 **NAME**

60697 shmdt — XSI shared memory detach operation

60698 **SYNOPSIS**

```
60699 XSI #include <sys/shm.h>
60700 int shmdt(const void *shmaddr);
```

60701 **DESCRIPTION**

60702 The *shmdt()* function operates on XSI shared memory (see XBD Section 3.340, on page 88). It is  
 60703 unspecified whether this function interoperates with the realtime interprocess communication  
 60704 facilities defined in Section 2.8 (on page 497).

60705 The *shmdt()* function detaches the shared memory segment located at the address specified by  
 60706 *shmaddr* from the address space of the calling process.

60707 **RETURN VALUE**

60708 Upon successful completion, *shmdt()* shall decrement the value of *shm\_nattch* in the data  
 60709 structure associated with the shared memory ID of the attached shared memory segment and  
 60710 return 0.

60711 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return -1, and *errno*  
 60712 shall be set to indicate the error.

60713 **ERRORS**

60714 The *shmdt()* function shall fail if:

60715	[EINVAL]	The value of <i>shmaddr</i> is not the data segment start address of a shared memory segment.
60716		

60717 **EXAMPLES**

60718 None.

60719 **APPLICATION USAGE**

60720 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 60721 Application developers who need to use IPC should design their applications so that modules  
 60722 using the IPC routines described in Section 2.7 (on page 496) can be easily modified to use the  
 60723 alternative interfaces.

60724 **RATIONALE**

60725 None.

60726 **FUTURE DIRECTIONS**

60727 None.

60728 **SEE ALSO**

60729 Section 2.7 (on page 496), Section 2.8 (on page 497), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*,  
 60730 *shmget()*, *shm\_open()*, *shm\_unlink()*

60731 XBD Section 3.340 (on page 88), *<sys/shm.h>*

60732 **CHANGE HISTORY**

60733 First released in Issue 2. Derived from Issue 2 of the SVID.

**shmdt()**

60734 **Issue 5**

60735 Moved from SHARED MEMORY to BASE.

60736 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
60737 DIRECTIONS to a new APPLICATION USAGE section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

60738 **NAME**

60739 shmget — get an XSI shared memory segment

60740 **SYNOPSIS**

```
60741 XSI #include <sys/shm.h>
60742 int shmget(key_t key, size_t size, int shmflg);
```

60743 **DESCRIPTION**

60744 The *shmget()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 88). It is  
 60745 unspecified whether this function interoperates with the realtime interprocess communication  
 60746 facilities defined in [Section 2.8](#) (on page 497).

60747 The *shmget()* function shall return the shared memory identifier associated with *key*.

60748 A shared memory identifier, associated data structure, and shared memory segment of at least  
 60749 *size* bytes (see [<sys/shm.h>](#)) are created for *key* if one of the following is true:

- 60750 • The argument *key* is equal to `IPC_PRIVATE`.
- 60751 • The argument *key* does not already have a shared memory identifier associated with it and  
 60752 (*shmflg* & `IPC_CREAT`) is non-zero.

60753 Upon creation, the data structure associated with the new shared memory identifier shall be  
 60754 initialized as follows:

- 60755 • The values of *shm\_perm.cuid*, *shm\_perm.uid*, *shm\_perm.cgid*, and *shm\_perm.gid* are set equal  
 60756 to the effective user ID and effective group ID, respectively, of the calling process.
- 60757 • The low-order nine bits of *shm\_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- 60758 • The value of *shm\_segsz* is set equal to the value of *size*.
- 60759 • The values of *shm\_lpid*, *shm\_nattch*, *shm\_atime*, and *shm\_dtime* are set equal to 0.
- 60760 • The value of *shm\_ctime* is set equal to the current time.

60761 When the shared memory segment is created, it shall be initialized with all zero values.

60762 **RETURN VALUE**

60763 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared  
 60764 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

60765 **ERRORS**

60766 The *shmget()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 60767 | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see <a href="#">Section 2.7</a> (on page 496). |
| 60768 |          |   |
| 60769 |          |   |
| 60770 | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but ( <i>shmflg</i> & <code>IPC_CREAT</code> ) && ( <i>shmflg</i> & <code>IPC_EXCL</code> ) is non-zero.                                  |
| 60771 |          |   |
| 60772 | [EINVAL] | A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum.   |
| 60773 |          |   |
| 60774 | [EINVAL] | No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0.           |
| 60775 |          |   |
| 60776 |          |   |

**shmget()**

60777	[ENOENT]	A shared memory identifier does not exist for the argument <i>key</i> and ( <i>shmflg</i> &IPC_CREAT) is 0.
60778		
60779	[ENOMEM]	A shared memory identifier and associated shared memory segment shall be created, but the amount of available physical memory is not sufficient to fill the request.
60780		
60781		
60782	[ENOSPC]	A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.
60783		
60784		

**EXAMPLES**

60785  
60786 None.

**APPLICATION USAGE**

60787 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
60788 Application developers who need to use IPC should design their applications so that modules  
60789 using the IPC routines described in Section 2.7 (on page 496) can be easily modified to use the  
60790 alternative interfaces.  
60791

**RATIONALE**

60792  
60793 None.

**FUTURE DIRECTIONS**

60794  
60795 None.

**SEE ALSO**

60796 Section 2.7 (on page 496), Section 2.8 (on page 497), *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*,  
60797 *shm\_unlink()*  
60798

60799 XBD Section 3.340 (on page 88), `<sys/shm.h>`

**CHANGE HISTORY**

60800 First released in Issue 2. Derived from Issue 2 of the SVID.

**Issue 5**

60802 Moved from SHARED MEMORY to BASE.

60804 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
60805 DIRECTIONS to a new APPLICATION USAGE section.

60806 **NAME**

60807 shutdown — shut down socket send and receive operations

60808 **SYNOPSIS**

```
60809 #include <sys/socket.h>
60810 int shutdown(int socket, int how);
```

60811 **DESCRIPTION**

60812 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket  
 60813 associated with the file descriptor *socket* to be shut down.

60814 The *shutdown()* function takes the following arguments:

60815	<i>socket</i>	Specifies the file descriptor of the socket.
60816	<i>how</i>	Specifies the type of shutdown. The values are as follows:
60817	SHUT_RD	Disables further receive operations.
60818	SHUT_WR	Disables further send operations.
60819	SHUT_RDWR	Disables further send and receive operations.

60820 The *shutdown()* function disables subsequent send and/or receive operations on a socket,  
 60821 depending on the value of the *how* argument.

60822 **RETURN VALUE**

60823 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*  
 60824 set to indicate the error.

60825 **ERRORS**

60826 The *shutdown()* function shall fail if:

60827	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
60828	[EINVAL]	The <i>how</i> argument is invalid.
60829	[ENOTCONN]	The socket is not connected.
60830	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.

60831 The *shutdown()* function may fail if:

60832	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
-------	-----------	---

60833 **EXAMPLES**

60834 None.

60835 **APPLICATION USAGE**

60836 None.

60837 **RATIONALE**

60838 None.

60839 **FUTURE DIRECTIONS**

60840 None.

60841 **SEE ALSO**

60842 [getsockopt\(\)](#), [pselect\(\)](#), [read\(\)](#), [recv\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#), [socket\(\)](#),  
 60843 [write\(\)](#)

60844 XBD [<sys/socket.h>](#)

## shutdown()

60845 **CHANGE HISTORY**

60846 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

60847 **NAME**

60848 sigaction — examine and change a signal action

60849 **SYNOPSIS**

```
60850 CX #include <signal.h>
60851 int sigaction(int sig, const struct sigaction *restrict act,
60852 struct sigaction *restrict oact);
```

60853 **DESCRIPTION**

60854 The *sigaction()* function allows the calling process to examine and/or specify the action to be  
 60855 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are  
 60856 defined in **<signal.h>**.

60857 The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>**  
 60858 header to include at least the following members:

Member Type	Member Name	Description
<b>void(*) (int)</b>	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
<b>sigset_t</b>	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
<b>int</b>	<i>sa_flags</i>	Special flags to affect behavior of signal.
<b>void(*) (int, siginfo_t *, void *)</b>	<i>sa_sigaction</i>	Pointer to a signal-catching function.

60868 The storage occupied by *sa\_handler* and *sa\_sigaction* may overlap, and a conforming application  
 60869 shall not use both simultaneously.

60870 If the argument *act* is not a null pointer, it points to a structure specifying the action to be  
 60871 associated with the specified signal. If the argument *oact* is not a null pointer, the action  
 60872 previously associated with the signal is stored in the location pointed to by the argument *oact*. If  
 60873 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to  
 60874 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall  
 60875 not be added to the signal mask using this mechanism; this restriction shall be enforced by the  
 60876 system without causing an error to be indicated.

60877 If the SA\_SIGINFO flag (see below) is cleared in the *sa\_flags* field of the **sigaction** structure, the  
 60878 *sa\_handler* field identifies the action to be associated with the specified signal. If the  
 60879 SA\_SIGINFO flag is set in the *sa\_flags* field, the *sa\_sigaction* field specifies a signal-catching  
 60880 function.

60881 The *sa\_flags* field can be used to modify the behavior of the specified signal.

60882 The following flags, defined in the **<signal.h>** header, can be set in *sa\_flags*:

60883 XSI SA\_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children  
 60884 continue.

60885 If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is not set in *sa\_flags*, and  
 60886 the implementation supports the SIGCHLD signal, then a SIGCHLD  
 60887 signal shall be generated for the calling process whenever any of its child  
 60888 XSI processes stop and a SIGCHLD signal may be generated for the calling  
 60889 process whenever any of its stopped child processes are continued. If *sig*  
 60890 is SIGCHLD and the SA\_NOCLDSTOP flag is set in *sa\_flags*, then the  
 60891 implementation shall not generate a SIGCHLD signal in this way.

**sigaction()**

60892	XSI	<b>SA_ONSTACK</b>	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
60893			
60894			
60895		<b>SA_RESETHAND</b>	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
60896			
60897			<b>Note:</b> SIGILL and SIGTRAP cannot be automatically reset when delivered.
60898			the system silently enforces this restriction.
60899			Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
60900			
60901			In addition, if this flag is set, <i>sigaction()</i> may behave as if the SA_NODEFER flag were also set.
60902			
60903		<b>SA_RESTART</b>	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
60904			
60905			
60906			
60907			
60908			
60909			
60910			
60911		<b>SA_SIGINFO</b>	If cleared and the signal is caught, the signal-catching function shall be entered as:
60912			
60913			<pre>void func(int signo);</pre>
60914			where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
60915			
60916			
60917			
60918			If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
60919			
60920			<pre>void func(int signo, siginfo_t *info, void *context);</pre>
60921			where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type <b>siginfo_t</b> explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type <b>ucontext_t</b> to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
60922			
60923			
60924			
60925			
60926			
60927			
60928			
60929			The <i>si_signo</i> member contains the system-generated signal number.
60930	XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
60931			
60932			
60933			The <i>si_code</i> member contains a code identifying the cause of the signal, as described in Section 2.4.3 (on page 486).
60934			

60935 SA\_NOCLDWAIT If set, and *sig* equals SIGCHLD, child processes of the calling processes  
60936 shall not be transformed into zombie processes when they terminate. If  
60937 the calling process subsequently waits for its children, and the process has  
60938 no unwaited-for children that were transformed into zombie processes, it  
60939 shall block until all of its children terminate, and *wait()*, *waitid()*, and  
60940 *waitpid()* shall fail and set *errno* to [ECHILD]. Otherwise, terminating  
60941 child processes shall be transformed into zombie processes, unless  
60942 SIGCHLD is set to SIG\_IGN.

60943 SA\_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask  
60944 on entry to the signal handler unless it is included in *sa\_mask*. Otherwise,  
60945 *sig* shall always be added to the thread's signal mask on entry to the  
60946 signal handler.

60947 When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask  
60948 is calculated and installed for the duration of the signal-catching function (or until a call to either  
60949 *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current  
60950 signal mask and the value of the *sa\_mask* for the signal being delivered, and unless  
60951 SA\_NODEFER or SA\_RESETHAND is set, then including the signal being delivered. If and  
60952 when the user's signal handler returns normally, the original signal mask is restored.

60953 Once an action is installed for a specific signal, it shall remain installed until another action is  
60954 explicitly requested (by another call to *sigaction()*), until the SA\_RESETHAND flag causes  
60955 resetting of the handler, or until one of the *exec* functions is called.

60956 If the previous action for *sig* had been established by *signal()*, the values of the fields returned in  
60957 the structure pointed to by *oact* are unspecified, and in particular *oact->sa\_handler* is not  
60958 necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a  
60959 copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the  
60960 signal shall be as if the original call to *signal()* were repeated.

60961 If *sigaction()* fails, no new signal handler is installed.

60962 It is unspecified whether an attempt to set the action for a signal that cannot be caught or  
60963 ignored to SIG\_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

60964 If SA\_SIGINFO is not set in *sa\_flags*, then the disposition of subsequent occurrences of *sig* when  
60965 it is already pending is implementation-defined; the signal-catching function shall be invoked  
60966 with a single argument. If SA\_SIGINFO is set in *sa\_flags*, then subsequent occurrences of *sig*  
60967 generated by *sigqueue()* or as a result of any signal-generating function that supports the  
60968 specification of an application-defined value (when *sig* is already pending) shall be queued in  
60969 FIFO order until delivered or accepted; the signal-catching function shall be invoked with three  
60970 arguments. The application specified value is passed to the signal-catching function as the  
60971 *si\_value* member of the **siginfo\_t** structure.

60972 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the  
60973 same signal is unspecified.

60974 **RETURN VALUE**

60975 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
60976 be set to indicate the error, and no new signal-catching function shall be installed.

**sigaction()**60977 **ERRORS**60978 The *sigaction()* function shall fail if:60979 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
60980 signal that cannot be caught or ignore a signal that cannot be ignored.60981 [ENOTSUP] The SA\_SIGINFO bit flag is set in the *sa\_flags* field of the **sigaction** structure.60982 The *sigaction()* function may fail if:60983 [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
60984 caught or ignored (or both).60985 In addition, the *sigaction()* function may fail if the SA\_SIGINFO flag is set in the *sa\_flags* field of  
60986 the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.60987 **EXAMPLES**60988 **Establishing a Signal Handler**60989 The following example demonstrates the use of *sigaction()* to establish a handler for the SIGINT  
60990 signal.

```

60991 #include <signal.h>
60992 static void handler(int signum)
60993 {
60994     /* Take appropriate actions for signal delivery */
60995 }
60996 int main()
60997 {
60998     struct sigaction sa;
60999     sa.sa_handler = handler;
61000     sigemptyset(&sa.sa_mask);
61001     sa.sa_flags = SA_RESTART; /* Restart functions if
61002                             interrupted by handler */
61003     if (sigaction(SIGINT, &sa, NULL) == -1)
61004         /* Handle error */;
61005     /* Further code */
61006 }

```

61007 **APPLICATION USAGE**

61008 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In  
61009 particular, *sigaction()* and *signal()* should not be used in the same process to control the same  
61010 signal. The behavior of async-signal-safe functions, as defined in their respective  
61011 DESCRIPTION sections, is as specified by this volume of POSIX.1-2008, regardless of invocation  
61012 from a signal-catching function. This is the only intended meaning of the statement that async-  
61013 signal-safe functions may be used in signal-catching functions without restrictions. Applications  
61014 must still consider all effects of such functions on such things as data structures, files, and  
61015 process state. In particular, application developers need to consider the restrictions on  
61016 interactions when interrupting *sleep()* and interactions among multiple handles for a file  
61017 description. The fact that any specific function is listed as async-signal-safe does not necessarily  
61018 mean that invocation of that function from a signal-catching function is recommended.

61019 In order to prevent errors arising from interrupting non-async-signal-safe function calls,  
61020 applications should protect calls to these functions either by blocking the appropriate signals or

61021 through the use of some programmatic semaphore (see *semget()*, *sem\_init()*, *sem\_open()*, and so  
 61022 on). Note in particular that even the “safe” functions may modify *errno*; the signal-catching  
 61023 function, if not executing as an independent thread, should save and restore its value in order to  
 61024 avoid the possibility that delivery of a signal in between an error return from a function that sets  
 61025 *errno* and the subsequent examination of *errno* could result in the signal-catching function  
 61026 changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of  
 61027 application routines and asynchronous data access. Note that *longjmp()* and *siglongjmp()* are not  
 61028 in the list of async-signal-safe functions. This is because the code executing after *longjmp()* and  
 61029 *siglongjmp()* can call any unsafe functions with the same danger as calling those unsafe  
 61030 functions directly from the signal handler. Applications that use *longjmp()* and *siglongjmp()* from  
 61031 within signal handlers require rigorous protection in order to be portable. Many of the other  
 61032 functions that are excluded from the list are traditionally implemented using either *malloc()* or  
 61033 *free()* functions or the standard I/O library, both of which traditionally use data structures in a  
 61034 non-async-signal-safe manner. Since any combination of different functions using a common  
 61035 data structure can cause async-signal-safety problems, this volume of POSIX.1-2008 does not  
 61036 define the behavior when any unsafe function is called in a signal handler that interrupts an  
 61037 unsafe function.

61038 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is  
 61039 undefined if the signal handler calls any function in the standard library other than one of the  
 61040 functions listed in the table of async-signal-safe functions in Section 2.4.3 (on page 486), or refers  
 61041 to any object other than *errno* with static storage duration other than by assigning a value to a  
 61042 static storage duration variable of type **volatile sig\_atomic\_t**. Unless all signal handlers have  
 61043 *errno* set on return as it was on entry, the value of *errno* is unspecified.

61044 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An  
 61045 alternate stack may be specified to receive a subset of the signals being caught.

61046 When the signal handler returns, the receiving thread resumes execution at the point it was  
 61047 interrupted unless the signal handler makes other arrangements. If *longjmp()* or *\_longjmp()* is  
 61048 used to leave the signal handler, then the signal mask must be explicitly restored.

61049 This volume of POSIX.1-2008 defines the third argument of a signal handling function when  
 61050 SA\_SIGINFO is set as a **void \*** instead of a **ucontext\_t \***, but without requiring type checking.  
 61051 New applications should explicitly cast the third argument of the signal handling function to  
 61052 **ucontext\_t \***.

61053 The BSD optional four argument signal handling function is not supported by this volume of  
 61054 POSIX.1-2008. The BSD declaration would be:

```
61055 void handler(int sig, int code, struct sigcontext *scp,  
61056             char *addr);
```

61057 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer  
 61058 to the **sigcontext** structure, and *addr* is additional address information. Much the same  
 61059 information is available in the objects pointed to by the second argument of the signal handler  
 61060 specified when SA\_SIGINFO is set.

61061 Since the *sigaction()* function is allowed but not required to set SA\_NODEFER when the  
 61062 application sets the SA\_RESETHAND flag, applications which depend on the SA\_RESETHAND  
 61063 functionality for the newly installed signal handler must always explicitly set SA\_NODEFER  
 61064 when they set SA\_RESETHAND in order to be portable.

**sigaction()**61065 **RATIONALE**

61066 Although this volume of POSIX.1-2008 requires that signals that cannot be ignored shall not be  
 61067 added to the signal mask when a signal-catching function is entered, there is no explicit  
 61068 requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact*  
 61069 argument. In other words, if SIGKILL is included in the *sa\_mask* field of *act*, it is unspecified  
 61070 whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa\_mask*  
 61071 field of *oact*.

61072 The SA\_NOCLDSTOP flag, when supplied in the *act->sa\_flags* parameter, allows overloading  
 61073 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated  
 61074 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching  
 61075 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on  
 61076 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of  
 61077 interest, the use of the SA\_NOCLDSTOP flag can prevent the overhead from invoking the  
 61078 signal-catching routine when they stop.

61079 Some historical implementations also define other mechanisms for stopping processes, such as  
 61080 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when  
 61081 processes stop due to this mechanism; however, that is beyond the scope of this volume of  
 61082 POSIX.1-2008.

61083 This volume of POSIX.1-2008 requires that calls to *sigaction()* that supply a NULL *act* argument  
 61084 succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or  
 61085 SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases  
 61086 and, in this respect, their behavior varies from *sigaction()*.

61087 This volume of POSIX.1-2008 requires that *sigaction()* properly save and restore a signal action  
 61088 set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is  
 61089 true, nor could there be given the greater amount of information conveyed by the **sigaction**  
 61090 structure. Because of this, applications should avoid using both functions for the same signal in  
 61091 the same process. Since this cannot always be avoided in case of general-purpose library  
 61092 routines, they should always be implemented with *sigaction()*.

61093 It was intended that the *signal()* function should be implementable as a library routine using  
 61094 *sigaction()*.

61095 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990  
 61096 standard to allow the application to request on a per-signal basis via an additional signal action  
 61097 flag that the extra parameters, including the application-defined signal value, if any, be passed to  
 61098 the signal-catching function.

61099 **FUTURE DIRECTIONS**

61100 None.

61101 **SEE ALSO**

61102 Section 2.4 (on page 484), *exec*, *kill()*, *\_longjmp()*, *longjmp()*, *pthread\_sigmask()*, *raise()*, *semget()*,  
 61103 *sem\_init()*, *sem\_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,  
 61104 *sigismember()*, *signal()*, *sigsuspend()*, *wait()*, *waitid()*

61105 XBD <**signal.h**>

61106 **CHANGE HISTORY**

61107 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61108 **Issue 5**

61109 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX  
61110 Threads Extension.

61111 In the DESCRIPTION, the second argument to *func* when SA\_SIGINFO is set is no longer  
61112 permitted to be NULL, and the description of permitted **siginfo\_t** contents is expanded by  
61113 reference to **<signal.h>**.

61114 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]  
61115 error is deleted.

61116 **Issue 6**

61117 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on  
61118 Other Functions”, a reference to *sigpending()* is added.

61119 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal  
61120 Effects on Other Functions” are moved to a separate section of this volume of POSIX.1-2008.

61121 Text describing functionality from the Realtime Signals Extension option is marked.

61122 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 61123 • The [ENOTSUP] error condition is added.

61124 The normative text is updated to avoid use of the term “must” for application requirements.

61125 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the  
61126 ISO/IEC 9899: 1999 standard.

61127 References to the *wait3()* function are removed.

61128 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an  
61129 extension over the ISO C standard.

61130 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table  
61131 describing the **sigaction** structure.

61132 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the  
61133 DESCRIPTION duplicated later in the same section.

61134 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the  
61135 DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread  
61136 rather than the process.

61137 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the  
61138 EXAMPLES section.

61139 **Issue 7**

61140 Austin Group Interpretation 1003.1-2001 #004 is applied.

61141 Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the  
61142 SA\_NODEFER flag with respect to the signal mask, and clarifying the SA\_RESTART flag for  
61143 interrupted functions which use timeouts.

61144 Austin Group Interpretation 1003.1-2001 #156 is applied.

61145 SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.

61146 SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement  
61147 that when the SA\_RESETHAND flag is set, *sigaction()* shall behave as if the SA\_NODEFER flag  
61148 were also set.

**sigaction()**

- 61149            Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 61150            The description of the *si\_code* member is replaced with a reference to [Section 2.4.3](#) (on page 486).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61151 **NAME**

61152 sigaddset — add a signal to a signal set

61153 **SYNOPSIS**

```
61154 CX #include <signal.h>
61155 int sigaddset(sigset_t *set, int signo);
```

61156 **DESCRIPTION**

61157 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed  
61158 to by *set*.

61159 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
61160 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
61161 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
61162 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
61163 *sigwaitinfo()*, the results are undefined.

61164 **RETURN VALUE**

61165 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*  
61166 to indicate the error.

61167 **ERRORS**

61168 The *sigaddset()* function may fail if:

61169 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

61170 **EXAMPLES**

61171 None.

61172 **APPLICATION USAGE**

61173 None.

61174 **RATIONALE**

61175 None.

61176 **FUTURE DIRECTIONS**

61177 None.

61178 **SEE ALSO**

61179 Section 2.4 (on page 484), *pthread\_sigmask()*, *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,  
61180 *sigismember()*, *sigpending()*, *sigsuspend()*

61181 XBD <signal.h>

61182 **CHANGE HISTORY**

61183 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61184 **Issue 5**

61185 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
61186 previous issues.

61187 **Issue 6**

61188 The normative text is updated to avoid use of the term “must” for application requirements.

61189 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
61190 extension over the ISO C standard.

**sigaltstack()**61191 **NAME**

61192 sigaltstack — set and get signal alternate stack context

61193 **SYNOPSIS**

```
61194 XSI #include <signal.h>
61195 int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

61196 **DESCRIPTION**

61197 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack  
 61198 for signal handlers for the current thread. Signals that have been explicitly declared to execute  
 61199 on the alternate stack shall be delivered on the alternate stack.

61200 If *ss* is not a null pointer, it points to a **stack\_t** structure that specifies the alternate signal stack  
 61201 that shall take effect upon return from *sigaltstack()*. The *ss\_flags* member specifies the new stack  
 61202 state. If it is set to *SS\_DISABLE*, the stack is disabled and *ss\_sp* and *ss\_size* are ignored.  
 61203 Otherwise, the stack shall be enabled, and the *ss\_sp* and *ss\_size* members specify the new address  
 61204 and size of the stack.

61205 The range of addresses starting at *ss\_sp* up to but not including *ss\_sp+ss\_size* is available to the  
 61206 implementation for use as the stack. This function makes no assumptions regarding which end  
 61207 is the stack base and in which direction the stack grows as items are pushed.

61208 If *oss* is not a null pointer, upon successful completion it shall point to a **stack\_t** structure that  
 61209 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss\_sp*  
 61210 and *ss\_size* members specify the address and size of that stack. The *ss\_flags* member specifies the  
 61211 stack's state, and may contain one of the following values:

61212 **SS\_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to  
 61213 modify the alternate signal stack while the process is executing on it fail. This  
 61214 flag shall not be modified by processes.

61215 **SS\_DISABLE** The alternate signal stack is currently disabled.

61216 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to  
 61217 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*  
 61218 is defined to be the minimum stack size for a signal handler. In computing an alternate stack  
 61219 size, a program should add that amount to its stack requirements to allow for the system  
 61220 implementation overhead. The constants *SS\_ONSTACK*, *SS\_DISABLE*, *SIGSTKSZ*, and  
 61221 *MINSIGSTKSZ* are defined in **<signal.h>**.

61222 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new  
 61223 process image.

61224 In some implementations, a signal (whether or not indicated to execute on the alternate stack)  
 61225 shall always execute on the alternate stack if it is delivered while another signal is being caught  
 61226 using the alternate stack.

61227 Use of this function by library threads that are not bound to kernel-scheduled entities results in  
 61228 undefined behavior.

61229 **RETURN VALUE**

61230 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return -1 and set *errno*  
 61231 to indicate the error.

61232 **ERRORS**61233 The *sigaltstack()* function shall fail if:61234 [EINVAL] The *ss* argument is not a null pointer, and the *ss\_flags* member pointed to by *ss*  
61235 contains flags other than *SS\_DISABLE*.61236 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

61237 [EPERM] An attempt was made to modify an active stack.

61238 **EXAMPLES**61239 **Allocating Memory for an Alternate Stack**

61240 The following example illustrates a method for allocating memory for an alternate stack.

```

61241 #include <signal.h>
61242 ...
61243 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
61244     /* Error return. */
61245     sigstk.ss_size = SIGSTKSZ;
61246     sigstk.ss_flags = 0;
61247     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
61248         perror("sigaltstack");

```

61249 **APPLICATION USAGE**61250 On some implementations, stack space is automatically extended as needed. On those  
61251 implementations, automatic extension is typically not available for an alternate stack. If the stack  
61252 overflows, the behavior is undefined.61253 **RATIONALE**

61254 None.

61255 **FUTURE DIRECTIONS**

61256 None.

61257 **SEE ALSO**61258 [Section 2.4](#) (on page 484), *exec*, *sigaction()*, *sigsetjmp()*61259 XBD [<signal.h>](#)61260 **CHANGE HISTORY**

61261 First released in Issue 4, Version 2.

61262 **Issue 5**

61263 Moved from X/OPEN UNIX extension to BASE.

61264 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in  
61265 previous issues.61266 **Issue 6**

61267 The normative text is updated to avoid use of the term “must” for application requirements.

61268 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the  
61269 ISO/IEC 9899:1999 standard.61270 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence  
61271 to include “for the current thread”.

**sigdelset()**61272 **NAME**

61273 sigdelset — delete a signal from a signal set

61274 **SYNOPSIS**

```
61275 CX #include <signal.h>
61276 int sigdelset(sigset_t *set, int signo);
```

61277 **DESCRIPTION**

61278 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set  
 61279 pointed to by *set*.

61280 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
 61281 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
 61282 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
 61283 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
 61284 *sigwaitinfo()*, the results are undefined.

61285 **RETURN VALUE**

61286 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*  
 61287 to indicate the error.

61288 **ERRORS**

61289 The *sigdelset()* function may fail if:

61290 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
 61291 number.

61292 **EXAMPLES**

61293 None.

61294 **APPLICATION USAGE**

61295 None.

61296 **RATIONALE**

61297 None.

61298 **FUTURE DIRECTIONS**

61299 None.

61300 **SEE ALSO**

61301 Section 2.4 (on page 484), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*,  
 61302 *sigismember()*, *sigpending()*, *sigsuspend()*

61303 XBD [<signal.h>](#)

61304 **CHANGE HISTORY**

61305 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61306 **Issue 5**

61307 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
 61308 previous issues.

61309 **Issue 6**

61310 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an  
 61311 extension over the ISO C standard.

61312 **NAME**

61313 sigemptyset — initialize and empty a signal set

61314 **SYNOPSIS**

```
61315 CX #include <signal.h>
61316 int sigemptyset(sigset_t *set);
```

61317 **DESCRIPTION**

61318 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined  
 61319 in POSIX.1-2008 are excluded.

61320 **RETURN VALUE**

61321 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set  
 61322 *errno* to indicate the error.

61323 **ERRORS**

61324 No errors are defined.

61325 **EXAMPLES**

61326 None.

61327 **APPLICATION USAGE**

61328 None.

61329 **RATIONALE**

61330 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or set)  
 61331 all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the  
 61332 structure, such as a version field, to permit binary-compatibility between releases where the size  
 61333 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any  
 61334 other use of the signal set, even if such use is read-only (for example, as an argument to  
 61335 *sigpending()*). This function is not intended for dynamic allocation.

61336 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or  
 61337 exclude) all the signals defined in this volume of POSIX.1-2008. Although it is outside the scope  
 61338 of this volume of POSIX.1-2008 to place this requirement on signals that are implemented as  
 61339 extensions, it is recommended that implementation-defined signals also be affected by these  
 61340 functions. However, there may be a good reason for a particular signal not to be affected. For  
 61341 example, blocking or ignoring an implementation-defined signal may have undesirable side-  
 61342 effects, whereas the default action for that signal is harmless. In such a case, it would be  
 61343 preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

61344 In early proposals there was no distinction between invalid and unsupported signals (the names  
 61345 of optional signals that were not supported by an implementation were not defined by that  
 61346 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.  
 61347 With that distinction, it is not necessary to require implementations of these functions to  
 61348 determine whether an optional signal is actually supported, as that could have a significant  
 61349 performance impact for little value. The error could have been required for invalid signals and  
 61350 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is  
 61351 optional in both cases.

61352 **FUTURE DIRECTIONS**

61353 None.

**sigemptyset()**

System Interfaces

61354 **SEE ALSO**

61355 [Section 2.4](#) (on page 484), [pthread\\_sigmask\(\)](#), [sigaction\(\)](#), [sigaddset\(\)](#), [sigdelset\(\)](#), [sigfillset\(\)](#),  
61356 [sigismember\(\)](#), [sigpending\(\)](#), [sigsuspend\(\)](#)

61357 XBD [<signal.h>](#)

61358 **CHANGE HISTORY**

61359 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61360 **Issue 6**

61361 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
61362 extension over the ISO C standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61363 **NAME**

61364 sigfillset — initialize and fill a signal set

61365 **SYNOPSIS**

```
61366 CX #include <signal.h>
61367 int sigfillset(sigset_t *set);
```

61368 **DESCRIPTION**

61369 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals  
 61370 defined in this volume of POSIX.1-2008 are included.

61371 **RETURN VALUE**

61372 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*  
 61373 to indicate the error.

61374 **ERRORS**

61375 No errors are defined.

61376 **EXAMPLES**

61377 None.

61378 **APPLICATION USAGE**

61379 None.

61380 **RATIONALE**

61381 Refer to *sigemptyset()* (on page 1927).

61382 **FUTURE DIRECTIONS**

61383 None.

61384 **SEE ALSO**

61385 Section 2.4 (on page 484), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*,  
 61386 *sigismember()*, *sigpending()*, *sigsuspend()*

61387 XBD <signal.h>

61388 **CHANGE HISTORY**

61389 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61390 **Issue 6**

61391 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
 61392 extension over the ISO C standard.

**sighold()**61393 **NAME**

61394 sighold, sigignore, sigpause, sigrelse, sigset — signal management

61395 **SYNOPSIS**

```

61396 OB XSI #include <signal.h>
61397 int sighold(int sig);
61398 int sigignore(int sig);
61399 int sigpause(int sig);
61400 int sigrelse(int sig);
61401 void (*sigset(int sig, void (*disp)(int)))(int);

```

61402 **DESCRIPTION**

61403 Use of any of these functions is unspecified in a multi-threaded process.

61404 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal management.  
61405

61406 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's disposition, which may be SIG\_DFL, SIG\_IGN, or the address of a signal handler. If *sigset()* is used, and *disp* is the address of a signal handler, the system shall add *sig* to the signal mask of the calling process before executing the signal handler; when the signal handler returns, the system shall restore the signal mask of the calling process to its state prior to the delivery of the signal. In addition, if *sigset()* is used, and *disp* is equal to SIG\_HOLD, *sig* shall be added to the signal mask of the calling process and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp* is not equal to SIG\_HOLD, *sig* shall be removed from the signal mask of the calling process.

61416 The *sighold()* function shall add *sig* to the signal mask of the calling process.61417 The *sigrelse()* function shall remove *sig* from the signal mask of the calling process.61418 The *sigignore()* function shall set the disposition of *sig* to SIG\_IGN.61419 The *sigpause()* function shall remove *sig* from the signal mask of the calling process and suspend the calling process until a signal is received. The *sigpause()* function shall restore the signal mask of the process to its original state before returning.  
61421

61422 If the action for the SIGCHLD signal is set to SIG\_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait()*, *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

61427 **RETURN VALUE**61428 Upon successful completion, *sigset()* shall return SIG\_HOLD if the signal had been blocked and the signal's previous disposition if it had not been blocked. Otherwise, SIG\_ERR shall be returned and *errno* set to indicate the error.  
61429  
6143061431 The *sigpause()* function shall suspend execution of the thread until a signal is received, whereupon it shall return -1 and set *errno* to [EINTR].  
6143261433 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.  
61434

**61435 ERRORS**

61436 These functions shall fail if:

61437 [EINVAL] The *sig* argument is an illegal signal number.

61438 The *sigset()* and *sigignore()* functions shall fail if:

61439 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a  
61440 signal that cannot be ignored.

**61441 EXAMPLES**

61442 None.

**61443 APPLICATION USAGE**

61444 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
61445 signals; new applications should use the *sigaction()* function instead of the obsolescent *sigset()*  
61446 function.

61447 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish  
61448 critical regions of code that require the delivery of a signal to be temporarily deferred. For  
61449 broader portability, the *pthread\_sigmask()* or *sigprocmask()* functions should be used instead of  
61450 the obsolescent *sighold()* and *sigrelse()* functions.

61451 For broader portability, the *sigsuspend()* function should be used instead of the obsolescent  
61452 *sigpause()* function.

**61453 RATIONALE**

61454 Each of these historic functions has a direct analog in the other functions which are required to  
61455 be per-thread and thread-safe (aside from *sigprocmask()*, which is replaced by *pthread\_sigmask()*).  
61456 The *sigset()* function can be implemented as a simple wrapper for *sigaction()*. The *sighold()*  
61457 function is equivalent to *sigprocmask()* or *pthread\_sigmask()* with SIG\_BLOCK set. The *sigignore()*  
61458 function is equivalent to *sigaction()* with SIG\_IGN set. The *sigpause()* function is equivalent to  
61459 *sigsuspend()*. The *sigrelse()* function is equivalent to *sigprocmask()* or *pthread\_sigmask()* with  
61460 SIG\_UNBLOCK set.

**61461 FUTURE DIRECTIONS**

61462 These functions may be removed in a future version.

**61463 SEE ALSO**

61464 Section 2.4 (on page 484), *exec*, *pause()*, *pthread\_sigmask()*, *sigaction()*, *signal()*, *sigsuspend()*,  
61465 *wait()*, *waitid()*.

61466 XBD <signal.h>

**61467 CHANGE HISTORY**

61468 First released in Issue 4, Version 2.

**61469 Issue 5**

61470 Moved from X/OPEN UNIX extension to BASE.

61471 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of  
61472 the process to its original state before returning.

61473 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
61474 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
61475 [EINTR].

**sighold()**61476 **Issue 6**

61477 The normative text is updated to avoid use of the term “must” for application requirements.

61478 References to the *wait3()* function are removed.

61479 The XSI functions are split out into their own reference page.

61480 **Issue 7**

61481 SD5-XSH-ERN-113 and SD5-XSH-ERN-42 are applied, marking these functions obsolescent and  
61482 updating the APPLICATION USAGE and RATIONALE sections.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61483 **NAME**61484 `siginterrupt` — allow signals to interrupt functions61485 **SYNOPSIS**

```
61486 OB XSI #include <signal.h>
61487 int siginterrupt(int sig, int flag);
```

61488 **DESCRIPTION**

61489 The `siginterrupt()` function shall change the restart behavior when a function is interrupted by  
 61490 the specified signal. The function `siginterrupt(sig, flag)` has an effect as if implemented as:

```
61491 int siginterrupt(int sig, int flag) {
61492     int ret;
61493     struct sigaction act;
61494
61495     (void) sigaction(sig, NULL, &act);
61496     if (flag)
61497         act.sa_flags &= ~SA_RESTART;
61498     else
61499         act.sa_flags |= SA_RESTART;
61500     ret = sigaction(sig, &act, NULL);
61501     return ret;
61502 }
```

61502 **RETURN VALUE**

61503 Upon successful completion, `siginterrupt()` shall return 0; otherwise, -1 shall be returned and  
 61504 `errno` set to indicate the error.

61505 **ERRORS**

61506 The `siginterrupt()` function shall fail if:

61507 [EINVAL] The `sig` argument is not a valid signal number.

61508 **EXAMPLES**

61509 None.

61510 **APPLICATION USAGE**

61511 The `siginterrupt()` function supports programs written to historical system interfaces.  
 61512 Applications should use the `sigaction()` with the SA\_RESTART flag instead of the obsolescent  
 61513 `siginterrupt()` function.

61514 **RATIONALE**

61515 None.

61516 **FUTURE DIRECTIONS**

61517 None.

61518 **SEE ALSO**

61519 [Section 2.4](#) (on page 484), `sigaction()`

61520 XBD `<signal.h>`

61521 **CHANGE HISTORY**

61522 First released in Issue 4, Version 2.

**siginterrupt()***System Interfaces*61523 **Issue 5**

61524 Moved from X/OPEN UNIX extension to BASE.

61525 **Issue 6**61526 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in  
61527 the sample implementation given in the DESCRIPTION.61528 **Issue 7**61529 The *siginterrupt()* function is marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61530 **NAME**

61531 sigismember — test for a signal in a signal set

61532 **SYNOPSIS**

```
61533 CX #include <signal.h>
61534 int sigismember(const sigset_t *set, int signo);
```

61535 **DESCRIPTION**

61536 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set  
 61537 pointed to by *set*.

61538 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
 61539 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
 61540 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
 61541 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
 61542 *sigwaitinfo()*, the results are undefined.

61543 **RETURN VALUE**

61544 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of  
 61545 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.

61546 **ERRORS**

61547 The *sigismember()* function may fail if:

61548 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
 61549 number.

61550 **EXAMPLES**

61551 None.

61552 **APPLICATION USAGE**

61553 None.

61554 **RATIONALE**

61555 None.

61556 **FUTURE DIRECTIONS**

61557 None.

61558 **SEE ALSO**

61559 Section 2.4 (on page 484), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,  
 61560 *sigemptyset()*, *sigpending()*, *sigsuspend()*

61561 XBD <signal.h>

61562 **CHANGE HISTORY**

61563 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61564 **Issue 5**

61565 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
 61566 previous issues.

61567 **Issue 6**

61568 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
 61569 extension over the ISO C standard.

**siglongjmp()**61570 **NAME**

61571 siglongjmp — non-local goto with signal handling

61572 **SYNOPSIS**

```
61573 CX #include <setjmp.h>
61574 void siglongjmp(sigjmp_buf env, int val);
```

61575 **DESCRIPTION**61576 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 61577 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 61578 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env*
- 61579 argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

61580 **RETURN VALUE**

61581 After *siglongjmp()* is completed, program execution shall continue as if the corresponding

61582 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function

61583 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

61584 **ERRORS**

61585 No errors are defined.

61586 **EXAMPLES**

61587 None.

61588 **APPLICATION USAGE**

61589 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant

61590 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

61591 **RATIONALE**

61592 None.

61593 **FUTURE DIRECTIONS**

61594 None.

61595 **SEE ALSO**61596 *longjmp()*, *pthread\_sigmask()*, *setjmp()*, *sigsetjmp()*, *sigsuspend()*61597 XBD **<setjmp.h>**61598 **CHANGE HISTORY**

61599 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

61600 **Issue 5**

61601 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61602 **Issue 6**61603 The DESCRIPTION is rewritten in terms of *longjmp()*.

61604 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an

61605 extension over the ISO C standard.

61606 **NAME**61607 `signal` — signal management61608 **SYNOPSIS**61609 `#include <signal.h>`61610 `void (*signal(int sig, void (*func)(int)))(int);`61611 **DESCRIPTION**

61612 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 61613 conflict between the requirements described here and the ISO C standard is unintentional. This  
 61614 volume of POSIX.1-2008 defers to the ISO C standard.

61615 Use of this function is unspecified in a multi-threaded process.

61616 The `signal()` function chooses one of three ways in which receipt of the signal number `sig` is to be  
 61617 subsequently handled. If the value of `func` is `SIG_DFL`, default handling for that signal shall  
 61618 occur. If the value of `func` is `SIG_IGN`, the signal shall be ignored. Otherwise, the application  
 61619 shall ensure that `func` points to a function to be called when that signal occurs. An invocation of  
 61620 such a function because of a signal, or (recursively) of any further functions called by that  
 61621 invocation (other than functions in the standard library), is called a “signal handler”.

61622 When a signal occurs, and `func` points to a function, it is implementation-defined whether the  
 61623 equivalent of a:

61624 `signal(sig, SIG_DFL);`

61625 is executed or the implementation prevents some implementation-defined set of signals (at least  
 61626 including `sig`) from occurring until the current signal handling has completed. (If the value of `sig`  
 61627 is `SIGILL`, the implementation may alternatively define that no action is taken.) Next the  
 61628 equivalent of:

61629 `(*func)(sig);`

61630 is executed. If and when the function returns, if the value of `sig` was `SIGFPE`, `SIGILL`, or  
 61631 `SIGSEGV` or any other implementation-defined value corresponding to a computational  
 61632 exception, the behavior is undefined. Otherwise, the program shall resume execution at the  
 61633 CX point it was interrupted. If the signal occurs as the result of calling the `abort()`, `raise()`, `kill()`,  
 61634 `pthread_kill()`, or `sigqueue()` function, the signal handler shall not call the `raise()` function.

61635 CX If the signal occurs other than as the result of calling `abort()`, `raise()`, `kill()`, `pthread_kill()`, or  
 61636 `sigqueue()`, the behavior is undefined if the signal handler refers to any object with static storage  
 61637 duration other than by assigning a value to an object declared as volatile `sig_atomic_t`, or if the  
 61638 signal handler calls any function in the standard library other than one of the functions listed in  
 61639 Section 2.4 (on page 484). Furthermore, if such a call fails, the value of `errno` is unspecified.

61640 At program start-up, the equivalent of:

61641 `signal(sig, SIG_IGN);`

61642 is executed for some signals, and the equivalent of:

61643 `signal(sig, SIG_DFL);`

61644 CX is executed for all other signals (see `exec`).

61645 **RETURN VALUE**

61646 If the request can be honored, `signal()` shall return the value of `func` for the most recent call to  
 61647 `signal()` for the specified signal `sig`. Otherwise, `SIG_ERR` shall be returned and a positive value  
 61648 shall be stored in `errno`.

**signal()**61649 **ERRORS**61650 The *signal()* function shall fail if:61651 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
61652 signal that cannot be caught or ignore a signal that cannot be ignored.61653 The *signal()* function may fail if:61654 CX [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
61655 caught or ignored (or both).61656 **EXAMPLES**

61657 None.

61658 **APPLICATION USAGE**61659 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
61660 signals; new applications should use *sigaction()* rather than *signal()*.61661 **RATIONALE**

61662 None.

61663 **FUTURE DIRECTIONS**

61664 None.

61665 **SEE ALSO**61666 Section 2.4 (on page 484), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*61667 XBD <[signal.h](#)>61668 **CHANGE HISTORY**

61669 First released in Issue 1. Derived from Issue 1 of the SVID.

61670 **Issue 5**

61671 Moved from X/OPEN UNIX extension to BASE.

61672 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of  
61673 the process to its original state before returning.61674 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
61675 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
61676 [EINTR].61677 **Issue 6**

61678 Extensions beyond the ISO C standard are marked.

61679 The normative text is updated to avoid use of the term “must” for application requirements.

61680 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

61681 References to the *wait3()* function are removed.61682 The *sighold()*, *sigignore()*, *sigelse()*, and *sigset()* functions are split out onto their own reference  
61683 page.

61684 **NAME**

61685 signbit — test sign

61686 **SYNOPSIS**

61687 #include &lt;math.h&gt;

61688 int signbit(real-floating x);

61689 **DESCRIPTION**

61690 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 61691 conflict between the requirements described here and the ISO C standard is unintentional. This  
 61692 volume of POSIX.1-2008 defers to the ISO C standard.

61693 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,  
 61694 zeros, and infinities have a sign bit.

61695 **RETURN VALUE**

61696 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is  
 61697 negative.

61698 **ERRORS**

61699 No errors are defined.

61700 **EXAMPLES**

61701 None.

61702 **APPLICATION USAGE**

61703 None.

61704 **RATIONALE**

61705 None.

61706 **FUTURE DIRECTIONS**

61707 None.

61708 **SEE ALSO**61709 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*

61710 XBD &lt;math.h&gt;

61711 **CHANGE HISTORY**

61712 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**signgam()**

System Interfaces

61713 **NAME**

61714 signgam — log gamma function

61715 **SYNOPSIS**

```
61716 XSI #include <math.h>  
61717 extern int signgam;
```

61718 **DESCRIPTION**61719 Refer to *lgamma()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61720 **NAME**

61721 sigpause — remove a signal from the signal mask and suspend the thread

61722 **SYNOPSIS**

```
61723 OB XSI #include <signal.h>  
61724 int sigpause(int sig);
```

61725 **DESCRIPTION**61726 Refer to *sighold()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sigpending()**61727 **NAME**

61728 sigpending — examine pending signals

61729 **SYNOPSIS**

```
61730 CX #include <signal.h>
61731 int sigpending(sigset_t *set);
```

61732 **DESCRIPTION**

61733 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of  
 61734 signals that are blocked from delivery to the calling thread and that are pending on the process  
 61735 or the calling thread.

61736 **RETURN VALUE**

61737 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and  
 61738 *errno* set to indicate the error.

61739 **ERRORS**

61740 No errors are defined.

61741 **EXAMPLES**

61742 None.

61743 **APPLICATION USAGE**

61744 None.

61745 **RATIONALE**

61746 None.

61747 **FUTURE DIRECTIONS**

61748 None.

61749 **SEE ALSO**61750 *exec*, *pthread\_sigmask()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*61751 XBD <[signal.h](#)>61752 **CHANGE HISTORY**

61753 First released in Issue 3.

61754 **Issue 5**

61755 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61756 **Issue 6**

61757 The SYNOPSIS is marked CX since the presence of this function in the <[signal.h](#)> header is an  
 61758 extension over the ISO C standard.

61759 **NAME**

61760 sigprocmask — examine and change blocked signals

61761 **SYNOPSIS**

```
61762 CX #include <signal.h>
61763 int sigprocmask(int how, const sigset_t *restrict set,
61764 sigset_t *restrict oset);
```

61765 **DESCRIPTION**61766 Refer to [pthread\\_sigmask\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sigqueue()**61767 **NAME**

61768 sigqueue — queue a signal to a process

61769 **SYNOPSIS**

```
61770 CX #include <signal.h>
61771 int sigqueue(pid_t pid, int signo, const union sigval value);
```

61772 **DESCRIPTION**

61773 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value  
 61774 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking  
 61775 is performed but no signal is actually sent. The null signal can be used to check the validity of  
 61776 *pid*.

61777 The conditions required for a process to have permission to queue a signal to another process are  
 61778 the same as for the *kill()* function.

61779 The *sigqueue()* function shall return immediately. If SA\_SIGINFO is set for *signo* and if the  
 61780 resources were available to queue the signal, the signal shall be queued and sent to the receiving  
 61781 process. If SA\_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving  
 61782 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this  
 61783 call.

61784 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked  
 61785 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*  
 61786 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the  
 61787 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the  
 61788 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.  
 61789 The selection order between realtime and non-realtime signals, or between multiple pending  
 61790 non-realtime signals, is unspecified.

61791 **RETURN VALUE**

61792 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*  
 61793 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set  
 61794 *errno* to indicate the error.

61795 **ERRORS**

61796 The *sigqueue()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 61797 | [EAGAIN] | No resources are available to queue the signal. The process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded. |
| 61798 |          |   |
| 61799 |          |   |
| 61800 | [EINVAL] | The value of the <i>signo</i> argument is an invalid or unsupported signal number.  |
| 61801 | [EPERM]  | The process does not have appropriate privileges to send the signal to the receiving process.   |
| 61802 |          |   |
| 61803 | [ESRCH]  | The process <i>pid</i> does not exist.  |

61804 **EXAMPLES**

61805 None.

61806 **APPLICATION USAGE**

61807 None.

61808 **RATIONALE**

61809 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another  
 61810 process, specifying the application-defined value. This is common practice in realtime  
 61811 applications on existing realtime systems. It was felt that specifying another function in the  
 61812 *sig...* name space already carved out for signals was preferable to extending the interface to  
 61813 *kill()*.

61814 Such a function became necessary when the put/get event function of the message queues was  
 61815 removed. It should be noted that the *sigqueue()* function implies reduced performance in a  
 61816 security-conscious implementation as the access permissions between the sender and receiver  
 61817 have to be checked on each send when the *pid* is resolved into a target process. Such access  
 61818 checks were necessary only at message queue open in the previous interface.

61819 The standard developers required that *sigqueue()* have the same semantics with respect to the  
 61820 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty  
 61821 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the  
 61822 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function  
 61823 queues a signal to a single process specified by the *pid* argument.

61824 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An  
 61825 explicit limit on the number of queued signals that a process could send was introduced. While  
 61826 the limit is “per-sender”, this volume of POSIX.1-2008 does not specify that the resources be part  
 61827 of the state of the sender. This would require either that the sender be maintained after exit until  
 61828 all signals that it had sent to other processes were handled or that all such signals that had not  
 61829 yet been acted upon be removed from the queue(s) of the receivers. This volume of  
 61830 POSIX.1-2008 does not preclude this behavior, but an implementation that allocated queuing  
 61831 resources from a system-wide pool (with per-sender limits) and that leaves queued signals  
 61832 pending after the sender exits is also permitted.

61833 **FUTURE DIRECTIONS**

61834 None.

61835 **SEE ALSO**61836 [Section 2.8.1](#) (on page 497)61837 XBD [<signal.h>](#)61838 **CHANGE HISTORY**

61839 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
 61840 POSIX Threads Extension.

61841 **Issue 6**61842 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

61843 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 61844 implementation does not support the Realtime Signals Extension option.

61845 **Issue 7**61846 The *sigqueue()* function is moved from the Realtime Signals Extension option to the Base.

**sigrelse()**61847 **NAME**

61848 sigrelse, sigset — signal management

61849 **SYNOPSIS**

```
61850 OB XSI #include <signal.h>
61851 int sigrelse(int sig);
61852 void (*sigset(int sig, void (*disp)(int)))(int);
```

61853 **DESCRIPTION**61854 Refer to *sighold()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61855 **NAME**

61856 sigsetjmp — set jump point for a non-local goto

61857 **SYNOPSIS**

```
61858 CX #include <setjmp.h>
61859 int sigsetjmp(sigjmp_buf env, int savemask);
```

61860 **DESCRIPTION**61861 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 61862 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 61863 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 61864 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

61866 **RETURN VALUE**

61867 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from  
 61868 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

61869 **ERRORS**

61870 No errors are defined.

61871 **EXAMPLES**

61872 None.

61873 **APPLICATION USAGE**

61874 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for  
 61875 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

61876 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified  
 61877 whether the signal mask is saved.

61878 **RATIONALE**

61879 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to  
 61880 permit implementors to recognize the name in the compiler and not implement an actual  
 61881 function. These same restrictions apply to the *sigsetjmp()* macro.

61882 There are processors that cannot easily support these calls, but this was not considered a  
 61883 sufficient reason to exclude them.

61884 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *\_setjmp()* and  
 61885 *\_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as  
 61886 *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal  
 61887 masks, while *\_setjmp()* and *\_longjmp()* do not. On System V Release 3 and in corresponding  
 61888 issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal  
 61889 masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to  
 61890 signal masks is not specified, and a new set of functions is defined instead.

61891 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching  
 61892 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts  
 61893 saved by other threads would be at best a questionable practice and were not considered worthy  
 61894 of standardization.

**sigsetjmp()**61895 **FUTURE DIRECTIONS**

61896 None.

61897 **SEE ALSO**61898 *pthread\_sigmask()*, *siglongjmp()*, *signal()*, *sigsuspend()*61899 XBD **<setjmp.h>**61900 **CHANGE HISTORY**

61901 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61902 **Issue 5**

61903 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61904 **Issue 6**61905 The DESCRIPTION is reworded in terms of *setjmp()*.61906 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

61907

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61908 **NAME**

61909 sigsuspend — wait for a signal

61910 **SYNOPSIS**

```
61911 CX #include <signal.h>
61912 int sigsuspend(const sigset_t *sigmask);
```

61913 **DESCRIPTION**

61914 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set  
 61915 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose  
 61916 action is either to execute a signal-catching function or to terminate the process. This shall not  
 61917 cause any other signals that may have been pending on the process to become pending on the  
 61918 thread.

61919 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to  
 61920 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching  
 61921 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*  
 61922 call.

61923 It is not possible to block signals that cannot be ignored. This is enforced by the system without  
 61924 causing an error to be indicated.

61925 **RETURN VALUE**

61926 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion  
 61927 return value. If a return occurs,  $-1$  shall be returned and *errno* set to indicate the error.

61928 **ERRORS**

61929 The *sigsuspend()* function shall fail if:

61930 [EINTR] A signal is caught by the calling process and control is returned from the  
 61931 signal-catching function.

61932 **EXAMPLES**

61933 None.

61934 **APPLICATION USAGE**

61935 Normally, at the beginning of a critical code section, a specified set of signals is blocked using  
 61936 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait  
 61937 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was  
 61938 returned by the *sigprocmask()* call.

61939 **RATIONALE**

61940 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers  
 61941 can install an additional cancellation handler which resets the signal mask to the expected value.

```
61942 void cleanup(void *arg)
61943 {
61944     sigset_t *ss = (sigset_t *) arg;
61945     pthread_sigmask(SIG_SETMASK, ss, NULL);
61946 }
61947 int call_sigsuspend(const sigset_t *mask)
61948 {
61949     sigset_t oldmask;
61950     int result;
61951     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
61952     pthread_cleanup_push(cleanup, &oldmask);
```

**sigsuspend()**

```
61953         result = sigsuspend(sigmask);
61954         pthread_cleanup_pop(0);
61955         return result;
61956     }
```

**61957 FUTURE DIRECTIONS**

61958 None.

**61959 SEE ALSO**

61960 [Section 2.4](#) (on page 484), [pause\(\)](#), [sigaction\(\)](#), [sigaddset\(\)](#), [sigdelset\(\)](#), [sigemptyset\(\)](#), [sigfillset\(\)](#)

61961 XBD [<signal.h>](#)

**61962 CHANGE HISTORY**

61963 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**61964 Issue 5**

61965 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

**61966 Issue 6**

61967 The text in the RETURN VALUE section has been changed from “suspends process execution”  
61968 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

61969 Text in the APPLICATION USAGE section has been replaced.

61970 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
61971 extension over the ISO C standard.

**61972 Issue 7**

61973 SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

61974 **NAME**

61975 sigtimedwait, sigwaitinfo — wait for queued signals

61976 **SYNOPSIS**

```

61977 CX #include <signal.h>
61978
61978 int sigtimedwait(const sigset_t *restrict set,
61979                 siginfo_t *restrict info,
61980                 const struct timespec *restrict timeout);
61981 int sigwaitinfo(const sigset_t *restrict set,
61982                 siginfo_t *restrict info);

```

61983 **DESCRIPTION**

61984 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals  
61985 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the  
61986 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-  
61987 valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall return  
61988 immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. If the  
61989 Monotonic Clock option is supported, the CLOCK\_MONOTONIC clock shall be used to  
61990 measure the time interval specified by the *timeout* argument.

61991 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of  
61992 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the  
61993 lowest numbered one. The selection order between realtime and non-realtime signals, or  
61994 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at  
61995 the time of the call, the calling thread shall be suspended until one or more signals in *set* become  
61996 pending or until it is interrupted by an unblocked, caught signal.

61997 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is  
61998 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function shall be equivalent to  
61999 *sigwait()*, except that the selected signal number shall be stored in the *si\_signo* member, and the  
62000 cause of the signal shall be stored in the *si\_code* member. If any value is queued to the selected  
62001 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the  
62002 value shall be stored in the *si\_value* member of *info*. The system resource used to queue the  
62003 signal shall be released and returned to the system for other use. If no value is queued, the  
62004 content of the *si\_value* member is undefined. If no further signals are queued for the selected  
62005 signal, the pending indication for that signal shall be reset.

62006 **RETURN VALUE**

62007 Upon successful completion (that is, one of the signals specified by *set* is pending or is  
62008 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,  
62009 the function shall return a value of -1 and set *errno* to indicate the error.

62010 **ERRORS**

62011 The *sigtimedwait()* function shall fail if:

62012 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.

62013 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

62014 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be  
62015 documented in system documentation whether this error causes these  
62016 functions to fail.

**sigtimedwait()**

62017 The *sigtimedwait()* function may also fail if:  
 62018 [EINVAL] The *timeout* argument specified a *tv\_nsec* value less than zero or greater than  
 62019 or equal to 1 000 million.  
 62020 An implementation should only check for this error if no signal is pending in *set* and it is  
 62021 necessary to wait.

**EXAMPLES**

62022 None.  
 62023

**APPLICATION USAGE**

62024 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application developers  
 62025 should note that this is inconsistent with other functions such as *pthread\_cond\_timedwait()* that  
 62026 return [ETIMEDOUT].  
 62027

**RATIONALE**

62028 Existing programming practice on realtime systems uses the ability to pause waiting for a  
 62029 selected set of events and handle the first event that occurs in-line instead of in a signal-handling  
 62030 function. This allows applications to be written in an event-directed style similar to a state  
 62031 machine. This style of programming is useful for largescale transaction processing in which the  
 62032 overall throughput of an application and the ability to clearly track states are more important  
 62033 than the ability to minimize the response time of individual event handling.  
 62034

62035 It is possible to construct a signal-waiting macro function out of the realtime signal function  
 62036 mechanism defined in this volume of POSIX.1-2008. However, such a macro has to include the  
 62037 definition of a generalized handler for all signals to be waited on. A significant portion of the  
 62038 overhead of handler processing can be avoided if the signal-waiting function is provided by the  
 62039 kernel. This volume of POSIX.1-2008 therefore provides two signal-waiting functions—one that  
 62040 waits indefinitely and one with a timeout—as part of the overall realtime signal function  
 62041 specification.

62042 The specification of a function with a timeout allows an application to be written that can be  
 62043 broken out of a wait after a set period of time if no event has occurred. It was argued that setting  
 62044 a timer event before the wait and recognizing the timer event in the wait would also implement  
 62045 the same functionality, but at a lower performance level. Because of the performance  
 62046 degradation associated with the user-level specification of a timer event and the subsequent  
 62047 cancellation of that timer event after the wait completes for a valid event, and the complexity  
 62048 associated with handling potential race conditions associated with the user-level method, the  
 62049 separate function has been included.

62050 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*  
 62051 function defined by this volume of POSIX.1-2008. The only difference is that *sigwaitinfo()* returns  
 62052 the queued signal value in the *value* argument. The return of the queued value is required so that  
 62053 applications can differentiate between multiple events queued to the same signal number.

62054 The two distinct functions are being maintained because some implementations may choose to  
 62055 implement the POSIX Threads Extension functions and not implement the queued signals  
 62056 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*  
 62057 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a  
 62058 macro on *sigwaitinfo()*.

62059 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns  
 62060 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed  
 62061 wait, and immediate return, and concerns regarding consistency with other functions where the  
 62062 conditional and timed waits were separate functions from the pure blocking function. The  
 62063 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a macro

62064 with a null pointer for *timeout*.

62065 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-  
62066 generated signals. One important question was how many threads that are suspended in a call  
62067 to a *sigwait*() function for a signal should return from the call when the signal is sent. Four  
62068 choices were considered:

- 62069 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 62070 2. One or more threads return.
- 62071 3. All waiting threads return.
- 62072 4. Exactly one thread returns.

62073 Prohibiting multiple calls to *sigwait*() for the same signal was felt to be overly restrictive. The  
62074 “one or more” behavior made implementation of conforming packages easy at the expense of  
62075 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait*() in  
62076 application code in order to achieve predictable behavior. There was concern that the “all  
62077 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU  
62078 resources by replicating the signals in the general case. Furthermore, no convincing examples  
62079 could be presented that delivery to all was either simpler or more powerful than delivery to one.

62080 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*  
62081 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 62082 • A multi-way signal wait can be built from the single-way wait.
- 62083 • Signals should only be handled by application-level code, as library routines cannot guess  
62084 what the application wants to do with signals generated for the entire process.
- 62085 • Applications can thus arrange for a single thread to wait for any given signal and call any  
62086 needed routines upon its arrival.

62087 In an application that is using signals for interprocess communication, signal processing is  
62088 typically done in one place. Alternatively, if the signal is being caught so that process cleanup  
62089 can be done, the signal handler thread can call separate process cleanup routines for each  
62090 portion of the application. Since the application main line started each portion of the application,  
62091 it is at the right abstraction level to tell each portion of the application to clean up.

62092 Certainly, there exist programming styles where it is logical to consider waiting for a single  
62093 signal in multiple threads. A simple *sigwait\_multiple*() routine can be constructed to achieve this  
62094 goal. A possible implementation would be to have each *sigwait\_multiple*() caller registered as  
62095 having expressed interest in a set of signals. The caller then waits on a thread-specific condition  
62096 variable. A single server thread calls a *sigwait*() function on the union of all registered signals.  
62097 When the *sigwait*() function returns, the appropriate state is set and condition variables are  
62098 broadcast. New *sigwait\_multiple*() callers may cause the pending *sigwait*() call to be canceled  
62099 and reissued in order to update the set of signals being waited for.

#### 62100 FUTURE DIRECTIONS

62101 None.

#### 62102 SEE ALSO

62103 [Section 2.8.1](#) (on page 497), [pause\(\)](#), [pthread\\_sigmask\(\)](#), [sigaction\(\)](#), [sigpending\(\)](#), [sigsuspend\(\)](#),  
62104 [sigwait\(\)](#)

62105 XBD [<signal.h>](#), [<time.h>](#)

**sigtimedwait()**62106 **CHANGE HISTORY**

62107 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
62108 POSIX Threads Extension.

62109 **Issue 6**

62110 These functions are marked as part of the Realtime Signals Extension option.

62111 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function  
62112 has been corrected so that the second argument is of type **siginfo\_t** \*.

62113 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
62114 implementation does not support the Realtime Signals Extension option.

62115 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
62116 CLOCK\_MONOTONIC clock, if supported, is used to measure timeout intervals.

62117 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment  
62118 with the ISO/IEC 9899:1999 standard.

62119 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the  
62120 RETURN VALUE section to that in the original base document ("An implementation should  
62121 only check for this error if no signal is pending in *set* and it is necessary to wait").

62122 **Issue 7**

62123 The *sigtimedwait()* and *sigwaitinfo()* functions are moved from the Realtime Signals Extension  
62124 option to the Base.

62125 **NAME**

62126 sigwait — wait for queued signals

62127 **SYNOPSIS**

```
62128 CX #include <signal.h>
62129 int sigwait(const sigset_t *restrict set, int *restrict sig);
```

62130 **DESCRIPTION**

62131 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's  
 62132 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to  
 62133 the call to *sigwait()* there are multiple pending instances of a single signal number, it is  
 62134 implementation-defined whether upon successful return there are any remaining pending  
 62135 signals for that signal number. If the implementation supports queued signals and there are  
 62136 multiple signals queued for the signal number selected, the first such queued signal shall cause a  
 62137 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the  
 62138 time of the call, the thread shall be suspended until one or more becomes pending. The signals  
 62139 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior  
 62140 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

62141 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these  
 62142 threads shall return from *sigwait()* with the signal number. If more than a single thread is  
 62143 blocked in *sigwait()* for a signal when that signal is generated for the process, it is unspecified  
 62144 which of the waiting threads returns from *sigwait()*. If the signal is generated for a specific  
 62145 thread, as by *pthread\_kill()*, only that thread shall return.

62146 Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it  
 62147 shall be the lowest numbered one. The selection order between realtime and non-realtime  
 62148 signals, or between multiple pending non-realtime signals, is unspecified.

62149 **RETURN VALUE**

62150 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the  
 62151 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to  
 62152 indicate the error.

62153 **ERRORS**

62154 The *sigwait()* function may fail if:

62155 [EINVAL] The *set* argument contains an invalid or unsupported signal number.

62156 **EXAMPLES**

62157 None.

62158 **APPLICATION USAGE**

62159 None.

62160 **RATIONALE**

62161 To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2008  
 62162 provides the *sigwait()* function. For most cases where a thread has to wait for a signal, the  
 62163 *sigwait()* function should be quite convenient, efficient, and adequate.

62164 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that  
 62165 could be used by threads. After some consideration, threads were allowed to use semaphores  
 62166 and *sem\_post()* was defined to be async-signal and async-cancel-safe.

62167 In summary, when it is necessary for code run in response to an asynchronous signal to notify a  
 62168 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation  
 62169 provides semaphores, they also can be used, either following *sigwait()* or from within a signal

**sigwait()**

62170 handling routine previously registered with *sigaction()*.

**62171 FUTURE DIRECTIONS**

62172 None.

**62173 SEE ALSO**

62174 [Section 2.4](#) (on page 484), [Section 2.8.1](#) (on page 497), [pause\(\)](#), [pthread\\_sigmask\(\)](#), [sigaction\(\)](#),  
62175 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigtimedwait\(\)](#)

62176 XBD [<signal.h>](#), [<time.h>](#)

**62177 CHANGE HISTORY**

62178 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
62179 POSIX Threads Extension.

**62180 Issue 6**

62181 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the  
62182 ISO/IEC 9899:1999 standard.

62183 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the  
62184 DESCRIPTION to state that if more than a single thread is blocked in *sigwait()*, it is unspecified  
62185 which of the waiting threads returns, and that if a signal is generated for a specific thread only  
62186 that thread shall return.

**62187 Issue 7**

62188 Functionality relating to the Realtime Signals Extension option is moved to the Base.

*System Interfaces***sigwaitinfo()**62189 **NAME**

62190 sigwaitinfo — wait for queued signals

62191 **SYNOPSIS**62192 `#include <signal.h>`62193 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);`62194 **DESCRIPTION**62195 Refer to *sigtimedwait()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sin()**62196 **NAME**62197 `sin, sinf, sinl` — sine function62198 **SYNOPSIS**

```
62199     #include <math.h>
62200     double sin(double x);
62201     float sinf(float x);
62202     long double sinl(long double x);
```

62203 **DESCRIPTION**

62204 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 62205 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62206 volume of POSIX.1-2008 defers to the ISO C standard.

62207 These functions shall compute the sine of their argument *x*, measured in radians.

62208 An application wishing to check for error situations should set *errno* to zero and call  
 62209 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 62210 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 62211 zero, an error has occurred.

62212 **RETURN VALUE**

62213 Upon successful completion, these functions shall return the sine of *x*.

62214 **MX** If *x* is NaN, a NaN shall be returned.

62215 If *x* is  $\pm 0$ , *x* shall be returned.

62216 If *x* is subnormal, a range error may occur and *x* should be returned.

62217 If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 62218 defined value shall be returned.

62219 **ERRORS**

62220 These functions shall fail if:

62221 **MX** **Domain Error** The *x* argument is  $\pm\text{Inf}$ .

62222 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62223 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62224 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 62225 shall be raised.

62226 These functions may fail if:

62227 **MX** **Range Error** The value of *x* is subnormal

62228 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62229 then *errno* shall be set to [ERANGE]. If the integer expression  
 62230 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 62231 floating-point exception shall be raised.

62232 **EXAMPLES**62233 **Taking the Sine of a 45-Degree Angle**

```

62234 #include <math.h>
62235 ...
62236 double radians = 45.0 * M_PI / 180;
62237 double result;
62238 ...
62239 result = sin(radians);

```

62240 **APPLICATION USAGE**

62241 These functions may lose accuracy when their argument is near a multiple of  $\pi$  or is far from 0.0.

62242 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
62243 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62244 **RATIONALE**

62245 None.

62246 **FUTURE DIRECTIONS**

62247 None.

62248 **SEE ALSO**

62249 *asin()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

62250 XBD Section 4.19 (on page 116), **<math.h>**

62251 **CHANGE HISTORY**

62252 First released in Issue 1. Derived from Issue 1 of the SVID.

62253 **Issue 5**

62254 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
62255 in previous issues.

62256 **Issue 6**

62257 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

62258 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
62259 revised to align with the ISO/IEC 9899:1999 standard.

62260 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
62261 marked.

**sinh()**62262 **NAME**62263 `sinh, sinhf, sinhl` — hyperbolic sine functions62264 **SYNOPSIS**

```
62265 #include <math.h>
62266 double sinh(double x);
62267 float sinhf(float x);
62268 long double sinhl(long double x);
```

62269 **DESCRIPTION**

62270 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62271 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62272 volume of POSIX.1-2008 defers to the ISO C standard.

62273 These functions shall compute the hyperbolic sine of their argument *x*.

62274 An application wishing to check for error situations should set *errno* to zero and call  
 62275 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 62276 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 62277 zero, an error has occurred.

62278 **RETURN VALUE**

62279 Upon successful completion, these functions shall return the hyperbolic sine of *x*.

62280 If the result would cause an overflow, a range error shall occur and  $\pm$ HUGE\_VAL,  
 62281  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as appropriate for  
 62282 the type of the function.

62283 MX If *x* is NaN, a NaN shall be returned.

62284 If *x* is  $\pm 0$  or  $\pm$ Inf, *x* shall be returned.

62285 If *x* is subnormal, a range error may occur and *x* should be returned.

62286 **ERRORS**

62287 These functions shall fail if:

62288 Range Error The result would cause an overflow.

62289 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62290 then *errno* shall be set to [ERANGE]. If the integer expression  
 62291 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 62292 floating-point exception shall be raised.

62293 These functions may fail if:

62294 MX Range Error The value *x* is subnormal.

62295 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62296 then *errno* shall be set to [ERANGE]. If the integer expression  
 62297 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 62298 floating-point exception shall be raised.

62299 **EXAMPLES**

62300 None.

62301 **APPLICATION USAGE**62302 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
62303 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.62304 **RATIONALE**

62305 None.

62306 **FUTURE DIRECTIONS**

62307 None.

62308 **SEE ALSO**62309 *asinh()*, *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*

62310 XBD Section 4.19 (on page 116), &lt;math.h&gt;

62311 **CHANGE HISTORY**

62312 First released in Issue 1. Derived from Issue 1 of the SVID.

62313 **Issue 5**62314 The DESCRIPTION is updated to indicate how an application should check for an error. This  
62315 text was previously published in the APPLICATION USAGE section.62316 **Issue 6**62317 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.62318 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
62319 revised to align with the ISO/IEC 9899:1999 standard.62320 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
62321 marked.

**sinl()**62322 **NAME**

62323           sinl — sine function

62324 **SYNOPSIS**

62325           #include &lt;math.h&gt;

62326           long double sinl(long double x);

62327 **DESCRIPTION**62328           Refer to *sin()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62329 **NAME**

62330 sleep — suspend execution for an interval of time

62331 **SYNOPSIS**

62332 #include &lt;unistd.h&gt;

62333 unsigned sleep(unsigned seconds);

62334 **DESCRIPTION**

62335 The *sleep()* function shall cause the calling thread to be suspended from execution until either  
 62336 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is  
 62337 delivered to the calling thread and its action is to invoke a signal-catching function or to  
 62338 terminate the process. The suspension time may be longer than requested due to the scheduling  
 62339 of other activity by the system.

62340 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the  
 62341 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*  
 62342 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 62343 unspecified whether it remains pending after *sleep()* returns or it is discarded.

62344 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a  
 62345 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 62346 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

62347 If a signal-catching function interrupts *sleep()* and examines or changes either the time a  
 62348 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 62349 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

62350 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 62351 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and  
 62352 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 62353 unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is  
 62354 restored as part of the environment.

62355 XSI Interactions between *sleep()* and *setitimer()* are unspecified.

62356 **RETURN VALUE**

62357 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*  
 62358 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested  
 62359 time minus the time actually slept) in seconds.

62360 **ERRORS**

62361 No errors are defined.

62362 **EXAMPLES**

62363 None.

62364 **APPLICATION USAGE**

62365 None.

62366 **RATIONALE**

62367 There are two general approaches to the implementation of the *sleep()* function. One is to use the  
 62368 *alarm()* function to schedule a SIGALRM signal and then suspend the calling thread waiting for  
 62369 that signal. The other is to implement an independent facility. This volume of POSIX.1-2008  
 62370 permits either approach.

62371 In order to comply with the requirement that no primitive shall change a process attribute unless  
 62372 explicitly described by this volume of POSIX.1-2008, an implementation using SIGALRM must  
 62373 carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the action

**sleep()**

62374 previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has  
 62375 been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to  
 62376 that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-  
 62377 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*  
 62378 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The  
 62379 action and blocking for SIGALRM must be saved and restored.

62380 Historical implementations often implement the SIGALRM-based version using *alarm()* and  
 62381 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*.  
 62382 Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid  
 62383 that window. That implementation introduces a different problem: when the SIGALRM signal  
 62384 interrupts a signal-catching function installed by the user to catch a different signal, the  
 62385 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,  
 62386 *alarm()*, and *sigsuspend()* can avoid these problems.

62387 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,  
 62388 differences between the two types of implementations. These are the cases mentioned in this  
 62389 volume of POSIX.1-2008 where some other activity relating to SIGALRM takes place, and the  
 62390 results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of  
 62391 concern to most applications.

62392 See also the discussion of the term *realtime* in *alarm()*.

62393 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early  
 62394 under *alarm()* applies to *sleep()* as well.

62395 Application developers should note that the type of the argument *seconds* and the return value of  
 62396 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application  
 62397 cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX}, which the  
 62398 ISO C standard sets as 65 535, and any application passing a larger value is restricting its  
 62399 portability. A different type was considered, but historical implementations, including those  
 62400 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

62401 Scheduling delays may cause the process to return from the *sleep()* function significantly after  
 62402 the requested time. In such cases, the return value should be set to zero, since the formula  
 62403 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an  
 62404 **unsigned**.

**FUTURE DIRECTIONS**

62405 None.

**SEE ALSO**

62408 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*

62409 XBD <**unistd.h**>

**CHANGE HISTORY**

62411 First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

62413 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

**Issue 6**

62414 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the  
 62415 RATIONALE section.  
 62416

62417 **NAME**62418        **snprintf** — print formatted output62419 **SYNOPSIS**

62420        #include &lt;stdio.h&gt;

62421        int snprintf(char \*restrict *s*, size\_t *n*,62422            const char \*restrict *format*, ...);62423 **DESCRIPTION**62424        Refer to *fprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**socketmark()**62425 **NAME**

62426 socketmark — determine whether a socket is at the out-of-band mark

62427 **SYNOPSIS**

```
62428 #include <sys/socket.h>
62429 int socketmark(int s);
```

62430 **DESCRIPTION**

62431 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at  
 62432 the out-of-band data mark (see [Section 2.10.12](#), on page 520). If the protocol for the socket  
 62433 supports out-of-band data by marking the stream with an out-of-band data mark, the  
 62434 *socketmark()* function shall return 1 when all data preceding the mark has been read and the out-  
 62435 of-band data mark is the first element in the receive queue. The *socketmark()* function shall not  
 62436 remove the mark from the stream.

62437 **RETURN VALUE**

62438 Upon successful completion, the *socketmark()* function shall return a value indicating whether  
 62439 the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data  
 62440 preceding the mark has been read, the return value shall be 1; if there is no mark, or if data  
 62441 precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it  
 62442 shall return a value of -1 and set *errno* to indicate the error.

62443 **ERRORS**62444 The *socketmark()* function shall fail if:

- 62445 [EBADF] The *s* argument is not a valid file descriptor.  
 62446 [ENOTTY] The file associated with the *s* argument is not a socket.

62447 **EXAMPLES**

62448 None.

62449 **APPLICATION USAGE**

62450 The use of this function between receive operations allows an application to determine which  
 62451 received data precedes the out-of-band data and which follows the out-of-band data.

62452 There is an inherent race condition in the use of this function. On an empty receive queue, the  
 62453 current read of the location might well be at the “mark”, but the system has no way of knowing  
 62454 that the next data segment that will arrive from the network will carry the mark, and  
 62455 *socketmark()* will return false, and the next read operation will silently consume the mark.

62456 Hence, this function can only be used reliably when the application already knows that the out-  
 62457 of-band data has been seen by the system or that it is known that there is data waiting to be read  
 62458 at the socket (via SIGURG or *select()*). See [Section 2.10.11](#) (on page 520), [Section 2.10.12](#) (on page  
 62459 520), [Section 2.10.14](#) (on page 521), and *pselect()* for details.

62460 **RATIONALE**

62461 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which  
 62462 implemented the same functionality on many implementations. Using a wrapper function  
 62463 follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other  
 62464 than those now included to support XSI STREAMS. The *socketmark()* function could be  
 62465 implemented as follows:

```
62466 #include <sys/ioctl.h>
62467 int socketmark(int s)
62468 {
62469     int val;
```

```
62470         if (ioctl(s, SIOCATMARK, &val) == -1)
62471             return(-1);
62472         return(val);
62473     }
```

62474 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of  
62475 SIOCATMARK.

62476 **FUTURE DIRECTIONS**

62477 None.

62478 **SEE ALSO**

62479 [Section 2.10.12](#) (on page 520), [pselect\(\)](#), [recv\(\)](#), [recvmsg\(\)](#)

62480 XBD [<sys/socket.h>](#)

62481 **CHANGE HISTORY**

62482 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

62483 **Issue 7**

62484 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**socket()**62485 **NAME**62486 `socket` — create an endpoint for communication62487 **SYNOPSIS**

```
62488 #include <sys/socket.h>
62489 int socket(int domain, int type, int protocol);
```

62490 **DESCRIPTION**

62491 The `socket()` function shall create an unbound socket in a communications domain, and return a  
 62492 file descriptor that can be used in later function calls that operate on sockets.

62493 The `socket()` function takes the following arguments:

62494	<i>domain</i>	Specifies the communications domain in which a socket is to be created.
62495	<i>type</i>	Specifies the type of socket to be created.
62496	<i>protocol</i>	Specifies a particular protocol to be used with the socket. Specifying a <i>protocol</i> 62497 of 0 causes <code>socket()</code> to use an unspecified default protocol appropriate for the 62498 requested socket type.

62499 The *domain* argument specifies the address family used in the communications domain. The  
 62500 address families supported by the system are implementation-defined.

62501 Symbolic constants that can be used for the domain argument are defined in the `<sys/socket.h>`  
 62502 header.

62503 The *type* argument specifies the socket type, which determines the semantics of communication  
 62504 over the socket. The following socket types are defined; implementations may specify additional  
 62505 socket types:

62506 `SOCK_STREAM` Provides sequenced, reliable, bidirectional, connection-mode byte streams,  
 62507 and may provide a transmission mechanism for out-of-band data.

62508 `SOCK_DGRAM` Provides datagrams, which are connectionless-mode, unreliable messages of  
 62509 fixed maximum length.

62510 `SOCK_SEQPACKET`

62511 Provides sequenced, reliable, bidirectional, connection-mode transmission  
 62512 paths for records. A record can be sent using one or more output operations  
 62513 and received using one or more input operations, but a single operation never  
 62514 transfers part of more than one record. Record boundaries are visible to the  
 62515 receiver via the `MSG_EOR` flag.

62516 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
 62517 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
 62518 be used. The protocols supported by the system are implementation-defined.

62519 The process may need to have appropriate privileges to use the `socket()` function or to create  
 62520 some sockets.

62521 **RETURN VALUE**

62522 Upon successful completion, `socket()` shall return a non-negative integer, the socket file  
 62523 descriptor. Otherwise, a value of `-1` shall be returned and `errno` set to indicate the error.

**62524 ERRORS**

62525 The *socket()* function shall fail if:

62526 [EAFNOSUPPORT]

62527 The implementation does not support the specified address family.

62528 [EMFILE] All file descriptors available to the process are currently open.

62529 [ENFILE] No more file descriptors are available for the system.

62530 [EPROTONOSUPPORT]

62531 The protocol is not supported by the address family, or the protocol is not supported by the implementation.

62533 [EPROTOTYPE] The socket type is not supported by the protocol.

62534 The *socket()* function may fail if:

62535 [EACCES] The process does not have appropriate privileges.

62536 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

62537 [ENOMEM] Insufficient memory was available to fulfill the request.

**62538 EXAMPLES**

62539 None.

**62540 APPLICATION USAGE**

62541 The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

62544 The application can determine whether an address family is supported by trying to create a socket with *domain* set to the protocol in question.

**62546 RATIONALE**

62547 None.

**62548 FUTURE DIRECTIONS**

62549 None.

**62550 SEE ALSO**

62551 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,  
62552 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*

62553 XBD <[netinet/in.h](#)>, <[sys/socket.h](#)>

**62554 CHANGE HISTORY**

62555 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**socketpair()**62556 **NAME**

62557 socketpair — create a pair of connected sockets

62558 **SYNOPSIS**

```
62559 #include <sys/socket.h>
62560 int socketpair(int domain, int type, int protocol,
62561               int socket_vector[2]);
```

62562 **DESCRIPTION**

62563 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,  
 62564 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two  
 62565 sockets shall be identical. The file descriptors used in referencing the created sockets shall be  
 62566 returned in *socket\_vector*[0] and *socket\_vector*[1].

62567 The *socketpair()* function takes the following arguments:

62568	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
62569	<i>type</i>	Specifies the type of sockets to be created.
62570	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a 62571 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol 62572 appropriate for the requested socket type.
62573	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

62574 The *type* argument specifies the socket type, which determines the semantics of communications  
 62575 over the socket. The following socket types are defined; implementations may specify additional  
 62576 socket types:

62577	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte 62578 streams, and may provide a transmission mechanism for out-of-band 62579 data.
62580	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages 62581 of fixed maximum length.
62582	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode transmission 62583 paths for records. A record can be sent using one or more output 62584 operations and received using one or more input operations, but a single 62585 operation never transfers part of more than one record. Record 62586 boundaries are visible to the receiver via the MSG_EOR flag.

62587 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
 62588 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
 62589 be used. The protocols supported by the system are implementation-defined.

62590 The process may need to have appropriate privileges to use the *socketpair()* function or to create  
 62591 some sockets.

62592 **RETURN VALUE**

62593 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and  
 62594 *errno* set to indicate the error.

62595 **ERRORS**

62596 The *socketpair()* function shall fail if:

62597	[EAFNOSUPPORT]	
62598		The implementation does not support the specified address family.

62599	[EMFILE]	All, or all but one, of the file descriptors available to the process are currently open.
62600		
62601	[ENFILE]	No more file descriptors are available for the system.
62602	[EOPNOTSUPP]	The specified protocol does not permit creation of socket pairs.
62603	[EPROTONOSUPPORT]	
62604		The protocol is not supported by the address family, or the protocol is not supported by the implementation.
62605		
62606	[EPROTOTYPE]	The socket type is not supported by the protocol.
62607		The <i>socketpair()</i> function may fail if:
62608	[EACCES]	The process does not have appropriate privileges.
62609	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
62610	[ENOMEM]	Insufficient memory was available to fulfill the request.
62611	<b>EXAMPLES</b>	
62612		None.
62613	<b>APPLICATION USAGE</b>	
62614		The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.
62615		
62616		
62617		The <i>socketpair()</i> function is used primarily with UNIX domain sockets and need not be supported for other domains.
62618		
62619	<b>RATIONALE</b>	
62620		None.
62621	<b>FUTURE DIRECTIONS</b>	
62622		None.
62623	<b>SEE ALSO</b>	
62624		<a href="#">socket()</a>
62625		XBD <a href="#">&lt;sys/socket.h&gt;</a>
62626	<b>CHANGE HISTORY</b>	
62627		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
62628	<b>Issue 7</b>	
62629		The description of the [EMFILE] error condition is aligned with the <i>pipe()</i> function.

**sprintf()***System Interfaces*62630 **NAME**62631 `printf` — print formatted output62632 **SYNOPSIS**62633 `#include <stdio.h>`62634 `int sprintf(char *restrict s, const char *restrict format, ...);`62635 **DESCRIPTION**62636 Refer to *fprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62637 **NAME**

62638 sqrt, sqrtf, sqrtl — square root function

62639 **SYNOPSIS**

```
62640 #include <math.h>
62641 double sqrt(double x);
62642 float sqrtf(float x);
62643 long double sqrtl(long double x);
```

62644 **DESCRIPTION**

62645 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62646 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62647 volume of POSIX.1-2008 defers to the ISO C standard.

62648 These functions shall compute the square root of their argument  $x$ ,  $\sqrt{x}$ .

62649 An application wishing to check for error situations should set *errno* to zero and call  
 62650 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 62651 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 62652 zero, an error has occurred.

62653 **RETURN VALUE**

62654 Upon successful completion, these functions shall return the square root of  $x$ .

62655 MX For finite values of  $x < -0$ , a domain error shall occur, and either a NaN (if supported), or an  
 62656 implementation-defined value shall be returned.

62657 MX If  $x$  is NaN, a NaN shall be returned.

62658 If  $x$  is  $\pm 0$  or  $+\text{Inf}$ ,  $x$  shall be returned.

62659 If  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 62660 defined value shall be returned.

62661 **ERRORS**

62662 These functions shall fail if:

62663 MX Domain Error The finite value of  $x$  is  $< -0$ , or  $x$  is  $-\text{Inf}$ .

62664 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62665 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62666 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 62667 shall be raised.

62668 **EXAMPLES**62669 **Taking the Square Root of 9.0**

```
62670 #include <math.h>
62671 ...
62672 double x = 9.0;
62673 double result;
62674 ...
62675 result = sqrt(x);
```

**sqrt()**62676 **APPLICATION USAGE**

62677 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
62678 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62679 **RATIONALE**

62680 None.

62681 **FUTURE DIRECTIONS**

62682 None.

62683 **SEE ALSO**

62684 *feclearexcept()*, *fetestexcept()*, *isnan()*

62685 XBD Section 4.19 (on page 116), `<math.h>`, `<stdio.h>`

62686 **CHANGE HISTORY**

62687 First released in Issue 1. Derived from Issue 1 of the SVID.

62688 **Issue 5**

62689 The DESCRIPTION is updated to indicate how an application should check for an error. This  
62690 text was previously published in the APPLICATION USAGE section.

62691 **Issue 6**

62692 The *sqrtf()* and *sqrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

62693 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
62694 revised to align with the ISO/IEC 9899:1999 standard.

62695 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
62696 marked.

*System Interfaces***srand()**62697 **NAME**

62698       srand — pseudo-random number generator

62699 **SYNOPSIS**

62700       #include &lt;stdlib.h&gt;

62701       void srand(unsigned seed);

62702 **DESCRIPTION**62703       Refer to *rand()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**srand48()***System Interfaces*62704 **NAME**

62705           srand48 — seed the uniformly distributed double-precision pseudo-random number generator

62706 **SYNOPSIS**

```
62707 XSI       #include <stdlib.h>  
62708       void srand48(long seedval);
```

62709 **DESCRIPTION**62710       Refer to *drand48()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***srandom()**62711 **NAME**

62712           srandom — seed pseudo-random number generator

62713 **SYNOPSIS**

```
62714 XSI       #include <stdlib.h>  
62715       void srandom(unsigned seed);
```

62716 **DESCRIPTION**62717       Refer to *initstate()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**sscanf()***System Interfaces*62718 **NAME**62719        **sscanf** — convert formatted input62720 **SYNOPSIS**

62721        #include &lt;stdio.h&gt;

62722        int sscanf(const char \*restrict *s*, const char \*restrict *format*, ...);62723 **DESCRIPTION**62724        Refer to *fscanf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***stat()**62725 **NAME**62726 `stat` — get file status62727 **SYNOPSIS**62728 `#include <sys/stat.h>`62729 `int stat(const char *restrict path, struct stat *restrict buf);`62730 **DESCRIPTION**62731 Refer to *fstatat()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**statvfs()***System Interfaces*62732 **NAME**

62733       statvfs — get file system information

62734 **SYNOPSIS**

62735       #include &lt;sys/statvfs.h&gt;

62736       int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);62737 **DESCRIPTION**62738       Refer to *fstatvfs()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62739 **NAME**

62740 stderr, stdin, stdout — standard I/O streams

62741 **SYNOPSIS**

62742 #include &lt;stdio.h&gt;

62743 extern FILE \*stderr, \*stdin, \*stdout;

62744 **DESCRIPTION**

62745 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62746 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62747 volume of POSIX.1-2008 defers to the ISO C standard.

62748 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type  
 62749 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer  
 62750 to designate the stream in all further transactions. Normally, there are three open streams with  
 62751 constant pointers declared in the <stdio.h> header and associated with the standard open files.

62752 At program start-up, three streams shall be predefined and need not be opened explicitly:  
 62753 *standard input* (for reading conventional input), *standard output* (for writing conventional output),  
 62754 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 62755 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 62756 the stream can be determined not to refer to an interactive device.

62757 CX The following symbolic values in <unistd.h> define the file descriptors that shall be associated  
 62758 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

62759 STDIN\_FILENO Standard input value, *stdin*. Its value is 0.62760 STDOUT\_FILENO Standard output value, *stdout*. Its value is 1.62761 STDERR\_FILENO Standard error value, *stderr*. Its value is 2.62762 The *stderr* stream is expected to be open for reading and writing.62763 **RETURN VALUE**

62764 None.

62765 **ERRORS**

62766 No errors are defined.

62767 **EXAMPLES**

62768 None.

62769 **APPLICATION USAGE**

62770 None.

62771 **RATIONALE**

62772 None.

62773 **FUTURE DIRECTIONS**

62774 None.

62775 **SEE ALSO**62776 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fprintf()*, *fread()*, *fscanf()*, *fseek()*, *getc()*, *gets()*, *popen()*,  
 62777 *putc()*, *puts()*, *read()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vfprintf()*

62778 XBD &lt;stdio.h&gt;, &lt;unistd.h&gt;

62779 **CHANGE HISTORY**

62780 First released in Issue 1.

62781 **Issue 6**

62782 Extensions beyond the ISO C standard are marked.

62783 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62784 **NAME**

62785 stpcpy — copy a string and return a pointer to the end of the result

62786 **SYNOPSIS**62787 CX `#include <string.h>`62788 `char *stpcpy(char *restrict s1, const char *restrict s2);`62789 **DESCRIPTION**62790 Refer to *strcpy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**stpncpy()**

System Interfaces

62791 **NAME**62792            `stpncpy` — copy fixed length string, returning a pointer to the array end62793 **SYNOPSIS**62794 CX        `#include <string.h>`62795            `char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);`62796 **DESCRIPTION**62797            Refer to *strncpy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**62798 NAME**

62799 `strcasecmp`, `strcasecmp_l`, `strncasecmp`, `strncasecmp_l` — case-insensitive string comparisons

**62800 SYNOPSIS**

```
62801 #include <strings.h>
62802 int strcasecmp(const char *s1, const char *s2);
62803 int strcasecmp_l(const char *s1, const char *s2,
62804                 locale_t locale);
62805 int strncasecmp(const char *s1, const char *s2, size_t n);
62806 int strncasecmp_l(const char *s1, const char *s2,
62807                  size_t n, locale_t locale);
```

**62808 DESCRIPTION**

62809 The `strcasecmp()` and `strcasecmp_l()` functions shall compare, while ignoring differences in case,  
 62810 the string pointed to by `s1` to the string pointed to by `s2`. The `strncasecmp()` and `strncasecmp_l()`  
 62811 functions shall compare, while ignoring differences in case, not more than `n` bytes from the  
 62812 string pointed to by `s1` to the string pointed to by `s2`.

62813 The `strcasecmp()` and `strncasecmp()` functions use the current locale of the process to determine  
 62814 the case of the characters.

62815 The `strcasecmp_l()` and `strncasecmp_l()` functions use the locale represented by `locale` to determine  
 62816 the case of the characters.

62817 When the `LC_CTYPE` category of the current locale is from the POSIX locale, `strcasecmp()` and  
 62818 `strncasecmp()` shall behave as if the strings had been converted to lowercase and then a byte  
 62819 comparison performed. Otherwise, the results are unspecified.

**62820 RETURN VALUE**

62821 Upon completion, `strcasecmp()` and `strcasecmp_l()` shall return an integer greater than, equal to,  
 62822 or less than 0, if the string pointed to by `s1` is, ignoring case, greater than, equal to, or less than  
 62823 the string pointed to by `s2`, respectively.

62824 Upon successful completion, `strncasecmp()` and `strncasecmp_l()` shall return an integer greater  
 62825 than, equal to, or less than 0, if the possibly null-terminated array pointed to by `s1` is, ignoring  
 62826 case, greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`,  
 62827 respectively.

**62828 ERRORS**

62829 The `strcasecmp_l()` and `strncasecmp_l()` functions may fail if:

62830 [EINVAL] `locale` is not a valid locale object handle.

**62831 EXAMPLES**

62832 None.

**62833 APPLICATION USAGE**

62834 None.

**62835 RATIONALE**

62836 None.

**62837 FUTURE DIRECTIONS**

62838 None.

**strcasemp()**62839 **SEE ALSO**62840 [wcscasemp\(\)](#)62841 XBD [<strings.h>](#)62842 **CHANGE HISTORY**

62843 First released in Issue 4, Version 2.

62844 **Issue 5**

62845 Moved from X/OPEN UNIX extension to BASE.

62846 **Issue 7**62847 The *strcasemp()* and *strncasemp()* functions are moved from the XSI option to the Base.62848 The *strcasemp\_l()* and *strncasemp\_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62850 **NAME**

62851 strcat — concatenate two strings

62852 **SYNOPSIS**

62853 #include &lt;string.h&gt;

62854 char \*strcat(char \*restrict s1, const char \*restrict s2);

62855 **DESCRIPTION**

62856 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62857 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62858 volume of POSIX.1-2008 defers to the ISO C standard.

62859 The *strcat()* function shall append a copy of the string pointed to by *s2* (including the  
 62860 terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2*  
 62861 overwrites the NUL character at the end of *s1*. If copying takes place between objects that  
 62862 overlap, the behavior is undefined.

62863 **RETURN VALUE**62864 The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.62865 **ERRORS**

62866 No errors are defined.

62867 **EXAMPLES**

62868 None.

62869 **APPLICATION USAGE**

62870 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3  
 62871 applications. Reliable error detection by this function was never guaranteed.

62872 **RATIONALE**

62873 None.

62874 **FUTURE DIRECTIONS**

62875 None.

62876 **SEE ALSO**62877 [strncat\(\)](#)62878 XBD [<string.h>](#)62879 **CHANGE HISTORY**

62880 First released in Issue 1. Derived from Issue 1 of the SVID.

62881 **Issue 6**62882 The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

**strchr()**62883 **NAME**

62884       strchr — string scanning operation

62885 **SYNOPSIS**

62886       #include &lt;string.h&gt;

62887       char \*strchr(const char \*s, int c);

62888 **DESCRIPTION**62889 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
62890 conflict between the requirements described here and the ISO C standard is unintentional. This  
62891 volume of POSIX.1-2008 defers to the ISO C standard.62892       The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string  
62893 pointed to by *s*. The terminating NUL character is considered to be part of the string.62894 **RETURN VALUE**62895       Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not  
62896 found.62897 **ERRORS**

62898       No errors are defined.

62899 **EXAMPLES**

62900       None.

62901 **APPLICATION USAGE**

62902       None.

62903 **RATIONALE**

62904       None.

62905 **FUTURE DIRECTIONS**

62906       None.

62907 **SEE ALSO**62908       [strchr\(\)](#)62909       XBD [<string.h>](#)62910 **CHANGE HISTORY**

62911       First released in Issue 1. Derived from Issue 1 of the SVID.

62912 **Issue 6**

62913       Extensions beyond the ISO C standard are marked.

62914 **NAME**

62915 strcmp — compare two strings

62916 **SYNOPSIS**

62917 #include &lt;string.h&gt;

62918 int strcmp(const char \*s1, const char \*s2);

62919 **DESCRIPTION**

62920 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62921 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62922 volume of POSIX.1-2008 defers to the ISO C standard.

62923 The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

62924 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 62925 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
 62926 being compared.

62927 **RETURN VALUE**

62928 Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the  
 62929 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,  
 62930 respectively.

62931 **ERRORS**

62932 No errors are defined.

62933 **EXAMPLES**62934 **Checking a Password Entry**

62935 The following example compares the information read from standard input to the value of the  
 62936 name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check  
 62937 will be made to see if the user entered the proper old password. The *crypt()* function shall  
 62938 encrypt the old password entered by the user, using the value of the encrypted password in the  
 62939 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the  
 62940 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program  
 62941 encrypts the new password so that it can store the information in the **passwd** structure.

```

62942 #include <string.h>
62943 #include <unistd.h>
62944 #include <stdio.h>
62945 ...
62946 int valid_change;
62947 struct passwd *p;
62948 char user[100];
62949 char oldpasswd[100];
62950 char newpasswd[100];
62951 char savepasswd[100];
62952 ...
62953 if (strcmp(p->pw_name, user) == 0) {
62954     if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
62955         strcpy(savepasswd, crypt(newpasswd, user));
62956         p->pw_passwd = savepasswd;
62957         valid_change = 1;
62958     }
62959     else {

```

**strcmp()**

```
62960         fprintf(stderr, "Old password is not valid\n");
62961     }
62962 }
62963 ...
```

**62964 APPLICATION USAGE**

62965 None.

**62966 RATIONALE**

62967 None.

**62968 FUTURE DIRECTIONS**

62969 None.

**62970 SEE ALSO**

62971 [strncmp\(\)](#)

62972 XBD [<string.h>](#)

**62973 CHANGE HISTORY**

62974 First released in Issue 1. Derived from Issue 1 of the SVID.

**62975 Issue 6**

62976 Extensions beyond the ISO C standard are marked.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62977 **NAME**62978 `strcoll, strcoll_l` — string comparison using collating information62979 **SYNOPSIS**

```
62980 #include <string.h>
62981 int strcoll(const char *s1, const char *s2);
62982 CX int strcoll_l(const char *s1, const char *s2,
62983 locale_t locale);
```

62984 **DESCRIPTION**

62985 CX For `strcoll()`: The functionality described on this reference page is aligned with the ISO C  
 62986 standard. Any conflict between the requirements described here and the ISO C standard is  
 62987 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

62988 CX The `strcoll()` and `strcoll_l()` functions shall compare the string pointed to by `s1` to the string  
 62989 pointed to by `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the current  
 62990 locale, or of the locale represented by `locale`, respectively.

62991 CX The `strcoll()` and `strcoll_l()` functions shall not change the setting of `errno` if successful.

62992 Since no return value is reserved to indicate an error, an application wishing to check for error  
 62993 CX situations should set `errno` to 0, then call `strcoll()`, or `strcoll_l()` then check `errno`.

62994 **RETURN VALUE**

62995 Upon successful completion, `strcoll()` shall return an integer greater than, equal to, or less than 0,  
 62996 according to whether the string pointed to by `s1` is greater than, equal to, or less than the string  
 62997 CX pointed to by `s2` when both are interpreted as appropriate to the current locale. On error,  
 62998 `strcoll()` may set `errno`, but no return value is reserved to indicate an error.

62999 Upon successful completion, `strcoll_l()` shall return an integer greater than, equal to, or less than  
 63000 0, according to whether the string pointed to by `s1` is greater than, equal to, or less than the  
 63001 string pointed to by `s2` when both are interpreted as appropriate to the locale represented by  
 63002 `locale`. On error, `strcoll_l()` may set `errno`, but no return value is reserved to indicate an error.

63003 **ERRORS**

63004 These functions may fail if:

63005 CX [EINVAL] The `s1` or `s2` arguments contain characters outside the domain of the collating  
 63006 sequence.

63007 The `strcoll_l()` function may fail if:

63008 CX [EINVAL] `locale` is not a valid locale object handle.

63009 **EXAMPLES**63010 **Comparing Nodes**

63011 The following example uses an application-defined function, `node_compare()`, to compare two  
 63012 nodes based on an alphabetical ordering of the `string` field.

```
63013 #include <string.h>
63014 ...
63015 struct node { /* These are stored in the table. */
63016     char *string;
63017     int length;
63018 };
63019 ...
```

**strcoll()**

```

63020     int node_compare(const void *node1, const void *node2)
63021     {
63022         return strcoll(((const struct node *)node1)->string,
63023                       ((const struct node *)node2)->string);
63024     }
63025     ...

```

**63026 APPLICATION USAGE**

63027 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

**63028 RATIONALE**

63029 None.

**63030 FUTURE DIRECTIONS**

63031 None.

**63032 SEE ALSO**

63033 *alphasort()*, *strcmp()*, *strxfrm()*

63034 XBD <*string.h*>

**63035 CHANGE HISTORY**

63036 First released in Issue 3.

**63037 Issue 5**

63038 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

**63039 Issue 6**

63040 Extensions beyond the ISO C standard are marked.

63041 The following new requirements on POSIX implementations derive from alignment with the  
 63042 Single UNIX Specification:

- 63043 • The [EINVAL] optional error condition is added.

63044 An example is added.

**63045 Issue 7**

63046 The *strcoll\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
 63047 Set Part 4.

63048 **NAME**

63049            strcpy, strcpy — copy a string and return a pointer to the end of the result

63050 **SYNOPSIS**

63051            #include &lt;string.h&gt;

63052 CX         char \*stpcpy(char \*restrict s1, const char \*restrict s2);

63053            char \*strcpy(char \*restrict s1, const char \*restrict s2);

63054 **DESCRIPTION**63055 CX         For *strcpy()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.63058 CX         The *stpcpy()* and *strcpy()* functions shall copy the string pointed to by *s2* (including the terminating NUL character) into the array pointed to by *s1*.

63060            If copying takes place between objects that overlap, the behavior is undefined.

63061 **RETURN VALUE**63062 CX         The *stpcpy()* function shall return a pointer to the terminating NUL character copied into the *s1* buffer.63064            The *strcpy()* function shall return *s1*.

63065            No return values are reserved to indicate an error.

63066 **ERRORS**

63067            No errors are defined.

63068 **EXAMPLES**63069            **Construction of a Multi-Part Message in a Single Buffer**

63070            #include &lt;string.h&gt;

63071            #include &lt;stdio.h&gt;

63072            int

63073            main (void)

63074            {

63075                 char buffer [10];

63076                 char \*name = buffer;

63077                 name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");

63078                 puts (buffer);

63079                 return 0;

63080            }

63081 **Initializing a String**63082            The following example copies the string "-----" into the *permstring* variable.

63083            #include &lt;string.h&gt;

63084            ...

63085            static char permstring[11];

63086            ...

63087            strcpy(permstring, "-----");

63088            ...

**strcpy()**63089 **Storing a Key and Data**

63090 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the  
 63091 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.  
 63092 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**  
 63093 **element** \*.)

```

63094 #include <string.h>
63095 #include <stdlib.h>
63096 #include <stdio.h>
63097 ...
63098 /* Structure used to read data and store it. */
63099 struct element {
63100     char *key;
63101     char *data;
63102 };
63103
63104 struct element *tbl, *curtbl;
63105 char *key, *data;
63106 int count;
63107 ...
63108 void dbfree(struct element *, int);
63109 ...
63110 if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
63111     perror("malloc"); dbfree(tbl, count); return NULL;
63112 }
63113 strcpy(curtbl->key, key);
63114
63115 if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
63116     perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
63117 }
63118 strcpy(curtbl->data, data);
63119 ...
  
```

63118 **APPLICATION USAGE**

63119 Character movement is performed differently in different implementations. Thus, overlapping  
 63120 moves may yield surprises.

63121 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3  
 63122 applications. Reliable error detection by this function was never guaranteed.

63123 **RATIONALE**

63124 None.

63125 **FUTURE DIRECTIONS**

63126 None.

63127 **SEE ALSO**

63128 *strncpy()*, *wscpy()*

63129 XBD **<string.h>**

63130 **CHANGE HISTORY**

63131 First released in Issue 1. Derived from Issue 1 of the SVID.

63132 **Issue 6**

63133 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

63134 **Issue 7**

63135 The *strcpy()* function is added from The Open Group Technical Standard, 2006, Extended API  
63136 Set Part 1.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**strcspn()**63137 **NAME**63138 `strcspn` — get the length of a complementary substring63139 **SYNOPSIS**63140 `#include <string.h>`63141 `size_t strcspn(const char *s1, const char *s2);`63142 **DESCRIPTION**63143 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
63144 conflict between the requirements described here and the ISO C standard is unintentional. This  
63145 volume of POSIX.1-2008 defers to the ISO C standard.63146 The `strcspn()` function shall compute the length (in bytes) of the maximum initial segment of the  
63147 string pointed to by `s1` which consists entirely of bytes *not* from the string pointed to by `s2`.63148 **RETURN VALUE**63149 The `strcspn()` function shall return the length of the computed segment of the string pointed to  
63150 by `s1`; no return value is reserved to indicate an error.63151 **ERRORS**

63152 No errors are defined.

63153 **EXAMPLES**

63154 None.

63155 **APPLICATION USAGE**

63156 None.

63157 **RATIONALE**

63158 None.

63159 **FUTURE DIRECTIONS**

63160 None.

63161 **SEE ALSO**63162 [strspn\(\)](#)63163 XBD [<string.h>](#)63164 **CHANGE HISTORY**

63165 First released in Issue 1. Derived from Issue 1 of the SVID.

63166 **Issue 5**63167 The RETURN VALUE section is updated to indicate that `strcspn()` returns the length of `s1`, and  
63168 not `s1` itself as was previously stated.63169 **Issue 6**63170 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is  
63171 updated to indicate that the computed segment length is returned, not the `s1` length.

63172 **NAME**63173            *strdup*, *strndup* — duplicate a specific number of bytes from a string63174 **SYNOPSIS**

```
63175 CX      #include <string.h>
63176          char *strdup(const char *s);
63177          char *strndup(const char *s, size_t size);
```

63178 **DESCRIPTION**

63179            The *strdup*() function shall return a pointer to a new string, which is a duplicate of the string  
 63180            pointed to by *s*. The returned pointer can be passed to *free*(). A null pointer is returned if the  
 63181            new string cannot be created.

63182            The *strndup*() function shall be equivalent to the *strdup*() function, duplicating the provided *s* in  
 63183            a new block of memory allocated as if by using *malloc*(), with the exception being that *strndup*()  
 63184            copies at most *size* plus one bytes into the newly allocated memory, terminating the new string  
 63185            with a NUL character. If the length of *s* is larger than *size*, only *size* bytes shall be duplicated. If  
 63186            *size* is larger than the length of *s*, all bytes in *s* shall be copied into the new memory buffer,  
 63187            including the terminating NUL character. The newly created string shall always be properly  
 63188            terminated.

63189 **RETURN VALUE**

63190            The *strdup*() function shall return a pointer to a new string on success. Otherwise, it shall return  
 63191            a null pointer and set *errno* to indicate the error.

63192            Upon successful completion, the *strndup*() function shall return a pointer to the newly allocated  
 63193            memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno*  
 63194            to indicate the error.

63195 **ERRORS**

63196            These functions shall fail if:

63197            [ENOMEM]       Storage space available is insufficient.

63198 **EXAMPLES**

63199            None.

63200 **APPLICATION USAGE**

63201            For functions that allocate memory as if by *malloc*(), the application should release such memory  
 63202            when it is no longer required by a call to *free*(). For *strdup*() and *strndup*(), this is the return  
 63203            value.

63204 **RATIONALE**

63205            None.

63206 **FUTURE DIRECTIONS**

63207            None.

63208 **SEE ALSO**

63209            *free*(), *wcsdup*()

63210            XBD <string.h>

63211 **CHANGE HISTORY**

63212            First released in Issue 4, Version 2.

**strdup()**63213 **Issue 5**

63214 Moved from X/OPEN UNIX extension to BASE.

63215 **Issue 7**

63216 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOMEM]  
63217 error to become a “shall fail” error.

63218 The *strdup()* function is moved from the XSI option to the Base.

63219 The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API  
63220 Set Part 1.

63221 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
63222 *malloc()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

63223 **NAME**63224 `strerror, strerror_l, strerror_r` — get error message string63225 **SYNOPSIS**63226 `#include <string.h>`63227 `char *strerror(int errnum);`63228 CX `char *strerror_l(int errnum, locale_t locale);`63229 `int strerror_r(int errnum, char *strerrbuf, size_t buflen);`63230 **DESCRIPTION**63231 CX For `strerror()`: The functionality described on this reference page is aligned with the ISO C  
63232 standard. Any conflict between the requirements described here and the ISO C standard is  
63233 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.63234 The `strerror()` function shall map the error number in `errnum` to a locale-dependent error  
63235 message string and shall return a pointer to it. Typically, the values for `errnum` come from `errno`,  
63236 but `strerror()` shall map any value of type `int` to a message.63237 The string pointed to shall not be modified by the application. The string may be overwritten by  
63238 a subsequent call to `strerror()`.63239 CX The string may be overwritten by a subsequent call to `strerror_l()` in the same thread.63240 The contents of the error message strings returned by `strerror()` should be determined by the  
63241 setting of the `LC_MESSAGES` category in the current locale.63242 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
63243 `strerror()`.63244 CX The `strerror()` and `strerror_l()` functions shall not change the setting of `errno` if successful.63245 Since no return value is reserved to indicate an error, an application wishing to check for error  
63246 situations should set `errno` to 0, then call `strerror()`, then check `errno`.63247 The `strerror()` function need not be thread-safe.63248 The `strerror_l()` function shall map the error number in `errnum` to a locale-dependent error  
63249 message string in the locale represented by `locale` and shall return a pointer to it.63250 The `strerror_r()` function shall map the error number in `errnum` to a locale-dependent error  
63251 message string and shall return the string in the buffer pointed to by `strerrbuf`, with length  
63252 `buflen`.63253 CX If the value of `errnum` is a valid error number, the message string shall indicate what error  
63254 occurred; otherwise, if these functions complete successfully, the message string shall indicate  
63255 that an unknown error occurred.63256 **RETURN VALUE**63257 Upon completion, whether successful or not, `strerror()` shall return a pointer to the generated  
63258 CX message string. On error `errno` may be set, but no return value is reserved to indicate an error.63259 Upon successful completion, `strerror_l()` shall return a pointer to the generated message string. If  
63260 `errnum` is not a valid error number, `errno` may be set to [EINVAL], but a pointer to a message  
63261 string shall still be returned. If any other error occurs, `errno` shall be set to indicate the error and  
63262 a null pointer shall be returned.63263 Upon successful completion, `strerror_r()` shall return 0. Otherwise, an error number shall be  
63264 returned to indicate the error.

**strerror()**63265 **ERRORS**

63266 These functions may fail if:

63267 CX [EINVAL] The value of *errnum* is not a valid error number.63268 The *strerror\_l()* function may fail if:63269 CX [EINVAL] The *locale* argument is not a valid locale object handle.63270 The *strerror\_r()* function may fail if:63271 CX [ERANGE] Insufficient storage was supplied via *strrdbuf* and *buflen* to contain the  
63272 generated message string.63273 **EXAMPLES**

63274 None.

63275 **APPLICATION USAGE**63276 Historically in some implementations, calls to *perror()* would overwrite the string that the  
63277 pointer returned by *strerror()* points to. Such implementations did not conform to the ISO C  
63278 standard; however, application developers should be aware of this behavior if they wish their  
63279 applications to be portable to such implementations.63280 **RATIONALE**63281 The *strerror\_l()* function is required to be thread-safe, thereby eliminating the need for an  
63282 equivalent to the *strerror\_r()* function.63283 Earlier versions of this standard did not explicitly require that the error message strings returned  
63284 by *strerror()* and *strerror\_r()* provide any information about the error. This version of the  
63285 standard requires a meaningful message for any successful completion.63286 Since no return value is reserved to indicate a *strerror()* error, but all calls (whether successful or  
63287 not) must return a pointer to a message string, on error *strerror()* can return a pointer to an  
63288 empty string or a pointer to a meaningful string that can be printed.63289 Note that the [EINVAL] error condition is a may fail error. If an invalid error number is supplied  
63290 as the value of *errnum*, applications should be prepared to handle any of the following:63291 1. Error (with no meaningful message): *errno* is set to [EINVAL], the return value is a pointer  
63292 to an empty string.63293 2. Successful completion: *errno* is unchanged and the return value points to a string like  
63294 "unknown error" or "error number xxx" (where *xxx* is the value of *errnum*).63295 3. Combination of #1 and #2: *errno* is set to [EINVAL] and the return value points to a string  
63296 like "unknown error" or "error number xxx" (where *xxx* is the value of *errnum*).63297 Since applications frequently use the return value of *strerror()* as an argument to  
63298 functions like *fprintf()* (without checking the return value) and since applications have no  
63299 way to parse an error message string to determine whether *errnum* represents a valid  
63300 error number, implementations are encouraged to implement #3. Similarly,  
63301 implementations are encouraged to have *strerror\_r()* return [EINVAL] and put a string  
63302 like "unknown error" or "error number xxx" in the buffer pointed to by *strrdbuf*  
63303 when the value of *errnum* is not a valid error number.63304 **FUTURE DIRECTIONS**

63305 None.

63306 **SEE ALSO**63307 *perror()*63308 XBD `<string.h>`63309 **CHANGE HISTORY**

63310 First released in Issue 3.

63311 **Issue 5**63312 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.63313 A note indicating that the *strerror()* function need not be reentrant is added to the  
63314 DESCRIPTION.63315 **Issue 6**

63316 Extensions beyond the ISO C standard are marked.

63317 The following new requirements on POSIX implementations derive from alignment with the  
63318 Single UNIX Specification:

- 63319 • In the RETURN VALUE section, the fact that *errno* may be set is added.
- 63320 • The [EINVAL] optional error condition is added.

63321 The normative text is updated to avoid use of the term “must” for application requirements.

63322 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.63323 The *strerror\_r()* function is marked as part of the Thread-Safe Functions option.63324 **Issue 7**

63325 Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

63326 Austin Group Interpretation 1003.1-2001 #156 is applied.

63327 Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the  
63328 generated error message is an empty string.

63329 SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section.

63330 The *strerror\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
63331 Set Part 4.63332 The *strerror\_r()* function is moved from the Thread-Safe Functions option to the Base.

**strfmon()**63333 **NAME**63334 `strfmon, strfmon_l` — convert monetary value to a string63335 **SYNOPSIS**

```
63336 #include <monetary.h>
63337
63337 ssize_t strfmon(char *restrict s, size_t maxsize,
63338               const char *restrict format, ...);
63339
63339 ssize_t strfmon_l(char *restrict s, size_t maxsize,
63340                 locale_t locale, const char *restrict format, ...);
```

63341 **DESCRIPTION**

63342 The `strfmon()` function shall place characters into the array pointed to by `s` as controlled by the  
 63343 string pointed to by `format`. No more than `maxsize` bytes are placed into the array.

63344 The format is a character string, beginning and ending in its initial state, if any, that contains two  
 63345 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*  
 63346 *specifications*, each of which shall result in the fetching of zero or more arguments which are  
 63347 converted and formatted. The results are undefined if there are insufficient arguments for the  
 63348 format. If the format is exhausted while arguments remain, the excess arguments are simply  
 63349 ignored.

63350 The application shall ensure that a conversion specification consists of the following sequence:

- 63351 • A '%' character
- 63352 • Optional flags
- 63353 • Optional field width
- 63354 • Optional left precision
- 63355 • Optional right precision
- 63356 • A required conversion specifier character that determines the conversion to be performed

63357 The `strfmon_l()` function shall be equivalent to the `strfmon()` function, except that the locale data  
 63358 used is from the locale represented by `locale`.

63359 **Flags**

63360 One or more of the following optional flags can be specified to control the conversion:

63361 `=f` An '=' followed by a single character `f` which is used as the numeric fill character. In  
 63362 order to work with precision or width counts, the fill character shall be a single byte  
 63363 character; if not, the behavior is undefined. The default numeric fill character is the  
 63364 <space>. This flag does not affect field width filling which always uses the <space>.  
 63365 This flag is ignored unless a left precision (see below) is specified.

63366 Do not format the currency amount with grouping characters. The default is to insert  
 63367 the grouping characters if defined for the current locale.

63368 + or ( Specify the style of representing positive and negative currency amounts. Only one of  
 63369 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'  
 63370 are used (for example, in many locales, the empty string if positive and '-' if  
 63371 negative). If '(' is specified, negative amounts are enclosed within parentheses. If  
 63372 neither flag is specified, the '+' style is used.

63373 ! Suppress the currency symbol from the output conversion.

63374 – Specify the alignment. If this flag is present the result of the conversion is left-justified  
 63375 (padded to the right) rather than right-justified. This flag shall be ignored unless a field  
 63376 width (see below) is specified.

#### 63377 Field Width

63378 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result  
 63379 of the conversion is right-justified (or left-justified if the flag '*-*' is specified). The  
 63380 default is 0.

#### 63381 Left Precision

63382 *#n* A '*#*' followed by a decimal digit string *n* specifying a maximum number of digits  
 63383 expected to be formatted to the left of the radix character. This option can be used to  
 63384 keep the formatted output from multiple calls to the *strfmon()* function aligned in the  
 63385 same columns. It can also be used to fill unused positions with a special character as in  
 63386 "*\$\*\*\*123.45*". This option causes an amount to be formatted as if it has the number  
 63387 of digits specified by *n*. If more than *n* digit positions are required, this conversion  
 63388 specification is ignored. Digit positions in excess of those actually required are filled  
 63389 with the numeric fill character (see the *=f* flag above).

63390 If grouping has not been suppressed with the '*^*' flag, and it is defined for the current  
 63391 locale, grouping separators are inserted before the fill characters (if any) are added.  
 63392 Grouping separators are not applied to fill characters even if the fill character is a digit.

63393 To ensure alignment, any characters appearing before or after the number in the  
 63394 formatted output such as currency or sign symbols are padded as necessary with  
 63395 *<space>* characters to make their positive and negative formats an equal length.

#### 63396 Right Precision

63397 *.p* A *<period>* followed by a decimal digit string *p* specifying the number of digits after  
 63398 the radix character. If the value of the right precision *p* is 0, no radix character appears.  
 63399 If a right precision is not included, a default specified by the current locale is used. The  
 63400 amount being formatted is rounded to the specified number of digits prior to  
 63401 formatting.

#### 63402 Conversion Specifier Characters

63403 The conversion specifier characters and their meanings are:

63404 *i* The **double** argument is formatted according to the locale's international currency  
 63405 format (for example, in the US: USD 1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the result  
 63406 of the conversion is unspecified.

63407 *n* The **double** argument is formatted according to the locale's national currency format  
 63408 (for example, in the US: \$1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the result of the  
 63409 conversion is unspecified.

63410 *%* Convert to a '*%*'; no argument is converted. The entire conversion specification shall  
 63411 be *%%*.

**strfmon()**63412 **Locale Information**

63413 The *LC\_MONETARY* category of the locale of the process affects the behavior of this function  
 63414 including the monetary radix character (which may be different from the numeric radix  
 63415 character affected by the *LC\_NUMERIC* category), the grouping separator, the currency symbols,  
 63416 and formats. The international currency symbol should be conformant with the ISO 4217:2001  
 63417 standard.

63418 If the value of *maxsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

63419 **RETURN VALUE**

63420 If the total number of resulting bytes including the terminating null byte is not more than  
 63421 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,  
 63422 not including the terminating NUL character. Otherwise, -1 shall be returned, the contents of the  
 63423 array are unspecified, and *errno* shall be set to indicate the error.

63424 **ERRORS**

63425 These functions shall fail if:

63426 [E2BIG] Conversion stopped due to lack of space in the buffer.

63427 The *strfmon\_l()* function may fail if:

63428 [EINVAL] *locale* is not a valid locale object.

63429 **EXAMPLES**

63430 Given a locale for the US and the values 123.45, -123.45, and 3456.781, the following output  
 63431 might be produced. Square brackets (" [ ] ") are used in this example to delimit the output.

63432	%n	[\$123.45]	Default formatting
63433		[-\$123.45]	
63434		[\$3,456.78]	
63435	%11n	[ \$123.45]	Right align within an 11-character field
63436		[-\$123.45]	
63437		[ \$3,456.78]	
63438	##5n	[ \$ 123.45]	Aligned columns for values up to 99999
63439		[-\$ 123.45]	
63440		[ \$ 3,456.78]	
63441	=%*#5n	[ \$***123.45]	Specify a fill character
63442		[-\$***123.45]	
63443		[ \$*3,456.78]	
63444	=%0#5n	[ \$000123.45]	Fill characters do not use grouping
63445		[-\$000123.45]	even if the fill character is a digit
63446		[ \$03,456.78]	
63447	%^#5n	[ \$ 123.45]	Disable the grouping separator
63448		[-\$ 123.45]	
63449		[ \$ 3456.78]	
63450	%^#5.0n	[ \$ 123]	Round off to whole units
63451		[-\$ 123]	
63452		[ \$ 3457]	
63453	%^#5.4n	[ \$ 123.4500]	Increase the precision
63454		[-\$ 123.4500]	
63455		[ \$ 3456.7810]	

63456	% (#5n	[ \$ 123.45 ]	Use an alternative pos/neg style
63457		[ (\$ 123.45) ]	
63458		[ \$ 3,456.78 ]	
63459	%! (#5n	[ 123.45 ]	Disable the currency symbol
63460		[ ( 123.45) ]	
63461		[ 3,456.78 ]	
63462	%-14#5.4n	[ \$ 123.4500 ]	Left-justify the output
63463		[-\$ 123.4500 ]	
63464		[ \$ 3,456.7810 ]	
63465	%14#5.4n	[ \$ 123.4500 ]	Corresponding right-justified output
63466		[ -\$ 123.4500 ]	
63467		[ \$ 3,456.7810 ]	

63468 See also the EXAMPLES section in *fprintf()*.

#### 63469 APPLICATION USAGE

63470 None.

#### 63471 RATIONALE

63472 None.

#### 63473 FUTURE DIRECTIONS

63474 Lowercase conversion characters are reserved for future standards use and uppercase for  
63475 implementation-defined use.

#### 63476 SEE ALSO

63477 *fprintf()*, *localeconv()*

63478 XBD <monetary.h>

#### 63479 CHANGE HISTORY

63480 First released in Issue 4.

#### 63481 Issue 5

63482 Moved from ENHANCED I18N to BASE.

63483 The [ENOSYS] error is removed.

63484 Text is added to the DESCRIPTION warning about values of *maxsize* that are greater than  
63485 {SSIZE\_MAX}.

#### 63486 Issue 6

63487 The normative text is updated to avoid use of the term “must” for application requirements.

63488 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the  
63489 ISO/IEC 9899:1999 standard.

63490 The EXAMPLES section is reworked, clarifying the output format.

#### 63491 Issue 7

63492 SD5-XSH-ERN-29 is applied, updating the examples for % (#5n and %! (#5n.

63493 SD5-XSH-ERN-233 is applied, changing the definition of the '+' or '(' flags to refer to  
63494 multiple locales.

63495 The *strfmon()* function is moved from the XSI option to the Base.

## strfmon()

63496  
63497

The *strfmon\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

63498 **NAME**63499 `strftime, strftime_l` — convert date and time to a string63500 **SYNOPSIS**

```
63501 #include <time.h>
63502 size_t strftime(char *restrict s, size_t maxsize,
63503               const char *restrict format, const struct tm *restrict timeptr);
63504 CX size_t strftime_l(char *restrict s, size_t maxsize,
63505                  const char *restrict format, const struct tm *restrict timeptr,
63506                  locale_t locale);
```

63507 **DESCRIPTION**

63508 CX For `strftime()`: The functionality described on this reference page is aligned with the ISO C  
 63509 standard. Any conflict between the requirements described here and the ISO C standard is  
 63510 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

63511 The `strftime()` function shall place bytes into the array pointed to by `s` as controlled by the string  
 63512 pointed to by `format`. The format is a character string, beginning and ending in its initial shift  
 63513 state, if any. The format string consists of zero or more conversion specifications and ordinary  
 63514 characters.

63515 Each conversion specification is introduced by the `'%'` character after which the following  
 63516 appear in sequence:

- 63517 CX
- An optional flag:
    - 63518 0 The zero character (`'0'`), which specifies that the character used as the padding
    - 63519 character is `'0'`,
    - 63520 + The `<plus-sign>` character (`'+'`), which specifies that the character used as the
    - 63521 padding character is `'0'`, and that if and only if the field being produced consumes
    - 63522 more than four bytes to represent a year (for `%F`, `%G`, or `%Y`) or more than two bytes to
    - 63523 represent the year divided by 100 (for `%C`) then a leading `<plus-sign>` character shall
    - 63524 be included if the year being processed is greater than or equal to zero or a leading
    - 63525 minus-sign character (`'-'`) shall be included if the year is less than zero.

63526 The default padding character is unspecified.

- An optional minimum field width. If the converted value, including any leading `'+'` or  
 63527 `'-'` sign, has fewer bytes than the minimum field width and the padding character is not  
 63528 the NUL character, the output shall be padded on the left (after any leading `'+'` or `'-'`  
 63529 sign) with the padding character.
- An optional `E` or `O` modifier.
- A terminating conversion specifier character that indicates the type of conversion to be  
 63532 applied.

63534 CX The results are unspecified if more than one flag character is specified, a flag character is  
 63535 specified without a minimum field width; a minimum field width is specified without a flag  
 63536 character; a modifier is specified with a flag or with a minimum field width; or if a minimum  
 63537 field width is specified for any conversion specifier other than `C`, `F`, `G`, or `Y`.

63538 All ordinary characters (including the terminating NUL character) are copied unchanged into  
 63539 the array. If copying takes place between objects that overlap, the behavior is undefined. No  
 63540 more than `maxsize` bytes are placed into the array. Each conversion specifier is replaced by  
 63541 appropriate characters as described in the following list. The appropriate characters are

**strftime()**

- 63542 determined using the *LC\_TIME* category of the current locale and by the values of zero or more  
 63543 members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the  
 63544 description. If any of the specified values are outside the normal range, the characters stored are  
 63545 unspecified.
- 63546 CX The *strftime\_l()* function shall be equivalent to the *strftime()* function, except that the locale data  
 63547 used is from the locale represented by *locale*.
- 63548 Local timezone information is used as though *strftime()* called *tzset()*.
- 63549 The following conversion specifiers shall be supported:
- 63550 a Replaced by the locale's abbreviated weekday name. [*tm\_wday*]
- 63551 A Replaced by the locale's full weekday name. [*tm\_wday*]
- 63552 b Replaced by the locale's abbreviated month name. [*tm\_mon*]
- 63553 B Replaced by the locale's full month name. [*tm\_mon*]
- 63554 c Replaced by the locale's appropriate date and time representation. (See the Base  
 63555 Definitions volume of POSIX.1-2008, <**time.h**>.)
- 63556 C Replaced by the year divided by 100 and truncated to an integer, as a decimal number.  
 63557 [*tm\_year*]
- 63558 If a minimum field width is not specified, the number of characters placed into the  
 63559 array pointed to by *s* will be the number of digits in the year divided by 100 or two,  
 63560 CX whichever is greater. If a minimum field width is specified, the number of characters  
 63561 placed into the array pointed to by *s* will be the number of digits in the year divided by  
 63562 100 or the minimum field width, whichever is greater.
- 63563 d Replaced by the day of the month as a decimal number [01,31]. [*tm\_mday*]
- 63564 D Equivalent to %m/%d/%y. [*tm\_mon, tm\_mday, tm\_year*]
- 63565 e Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded  
 63566 by a space. [*tm\_mday*]
- 63567 CX F Equivalent to %+4Y-%m-%d if no flag and no minimum field width are specified.  
 63568 [*tm\_year, tm\_mon, tm\_mday*]
- 63569 CX If a minimum field width of *x* is specified, the year shall be output as if by the *Y*  
 63570 specifier (described below) with whatever flag was given and a minimum field width  
 63571 of *x*-6. If *x* is less than 6, the behavior shall be as if *x* equalled 6.
- 63572 If the minimum field width is specified to be 10, and the year is four digits long, then  
 63573 the output string produced will match the ISO 8601:2004 standard subclause 4.1.2.2  
 63574 complete representation, extended format date representation of a specific day. If a +  
 63575 flag is specified, a minimum field width of *x* is specified, and *x*-7 bytes are sufficient to  
 63576 hold the digits of the year (not including any needed sign character), then the output  
 63577 will match the ISO 8601:2004 standard subclause 4.1.2.4 complete representation,  
 63578 expanded format date representation of a specific day.
- 63579 g Replaced by the last 2 digits of the week-based year (see below) as a decimal number  
 63580 [00,99]. [*tm\_year, tm\_wday, tm\_yday*]
- 63581 G Replaced by the week-based year (see below) as a decimal number (for example, 1977).  
 63582 [*tm\_year, tm\_wday, tm\_yday*]

63583	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
63584		
63585		
63586	h	Equivalent to %b. [ <i>tm_mon</i> ]
63587	H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [ <i>tm_hour</i> ]
63588	I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [ <i>tm_hour</i> ]
63589	j	Replaced by the day of the year as a decimal number [001,366]. [ <i>tm_yday</i> ]
63590	m	Replaced by the month as a decimal number [01,12]. [ <i>tm_mon</i> ]
63591	M	Replaced by the minute as a decimal number [00,59]. [ <i>tm_min</i> ]
63592	n	Replaced by a <newline>.
63593	p	Replaced by the locale's equivalent of either a.m. or p.m. [ <i>tm_hour</i> ]
63594	CX	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be equivalent to %I:%M:%S %p. [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]
63595		
63596	R	Replaced by the time in 24-hour notation (%H:%M). [ <i>tm_hour</i> , <i>tm_min</i> ]
63597	S	Replaced by the second as a decimal number [00,60]. [ <i>tm_sec</i> ]
63598	t	Replaced by a <tab>.
63599	T	Replaced by the time (%H:%M:%S). [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]
63600	u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [ <i>tm_wday</i> ]
63601		
63602	U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
63603		
63604		
63605	V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
63606		
63607		
63608		
63609		
63610	w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [ <i>tm_wday</i> ]
63611		
63612	W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
63613		
63614		
63615	x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2008, <time.h>.)
63616		
63617	X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2008, <time.h>.)
63618		
63619	y	Replaced by the last two digits of the year as a decimal number [00,99]. [ <i>tm_year</i> ]
63620	Y	Replaced by the year as a decimal number (for example, 1997). [ <i>tm_year</i> ]
63621	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the
63622		

**strftime()**

63623		year, or the minimum field width, whichever is greater.
63624	z	Replaced by the offset from UTC in the ISO 8601:2004 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [ <i>tm_isdst</i> ]
63625		
63626	CX	
63627		
63628		
63629	Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [ <i>tm_isdst</i> ]
63630		
63631	%	Replaced by %.
63632		If a conversion specification does not correspond to any of the above, the behavior is undefined.
63633	CX	If a <b>struct tm</b> broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z <i>strftime()</i> conversion specifiers are undefined, when <i>strftime()</i> is called with such a broken-down time structure.
63634		
63635		
63636		
63637		If a <b>struct tm</b> broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the current local timezone, when <i>strftime()</i> is called with such a broken-down time structure.
63638		
63639		
63640		<b>Modified Conversion Specifiers</b>
63641		Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in XBD Section 7.3.5, on page 158), the behavior shall be as if the unmodified conversion specification were used.
63642		
63643		
63644		
63645		
63646	%Ec	Replaced by the locale's alternative appropriate date and time representation.
63647	%EC	Replaced by the name of the base year (period) in the locale's alternative representation.
63648		
63649	%Ex	Replaced by the locale's alternative date representation.
63650	%EX	Replaced by the locale's alternative time representation.
63651	%Ey	Replaced by the offset from %EC (year only) in the locale's alternative representation.
63652	%EY	Replaced by the full alternative year representation.
63653	%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.
63654		
63655		
63656	%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.
63657		
63658	%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
63659	%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
63660	%Om	Replaced by the month using the locale's alternative numeric symbols.
63661	%OM	Replaced by the minutes using the locale's alternative numeric symbols.

63662	%OS	Replaced by the seconds using the locale's alternative numeric symbols.
63663	%Ou	Replaced by the weekday as a number in the locale's alternative representation (Monday=1).
63664		
63665	%OU	Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.
63666		
63667	%OV	Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.
63668		
63669	%Ow	Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
63670		
63671	%OW	Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
63672		
63673	%Oy	Replaced by the year (offset from %C) using the locale's alternative numeric symbols.
63674	%g, %G, and %V	give values according to the ISO 8601:2004 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.
63675		
63676		
63677		
63678		
63679		
63680		
63681		
63682		If a conversion specifier is not one of the above, the behavior is undefined.

**63683 RETURN VALUE**

63684 If the total number of resulting bytes including the terminating null byte is not more than  
 63685 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,  
 63686 not including the terminating NUL character. Otherwise, 0 shall be returned and the contents of  
 63687 the array are unspecified.

**63688 ERRORS**

63689 The *strftime\_l()* function may fail if:

63690 CX [EINVAL] *locale* is not a valid locale object handle.

**63691 EXAMPLES****63692 Getting a Localized Date String**

63693 The following example first sets the locale to the user's default. The locale information will be  
 63694 used in the *nl\_langinfo()* and *strftime()* functions. The *nl\_langinfo()* function returns the localized  
 63695 date string which specifies how the date is laid out. The *strftime()* function takes this  
 63696 information and, using the **tm** structure for values, places the date and time information into  
 63697 *datestring*.

```
63698 #include <time.h>
63699 #include <locale.h>
63700 #include <langinfo.h>
63701 ...
63702 struct tm *tm;
63703 char datestring[256];
63704 ...
63705 setlocale (LC_ALL, "");
```

**strftime()**

```
63706     ...
63707     strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
63708     ...
```

**63709 APPLICATION USAGE**

63710 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

63711 Some of the conversion specifications are duplicates of others. They are included for  
63712 compatibility with *nl\_cxtime()* and *nl\_ascxtime()*, which were published in Issue 2.

63713 The %C, %F, %G, and %Y format specifiers in *strftime()* always print full values, but the *strptime()*  
63714 %C, %F, and %Y format specifiers only scan two digits (assumed to be the first two digits of a  
63715 four-digit year) for %C and four digits (assumed to be the entire (four-digit) year) for %F and %Y.  
63716 This mimics the behavior of *printf()* and *scanf()*; that is:

```
63717 printf("%2d", x = 1000);
```

63718 prints "1000", but:

```
63719 scanf("%2d", &x);
```

63720 when given "1000" as input will only store 10 in *x*). Applications using extended ranges of  
63721 years must be sure that the number of digits specified for scanning years with *strptime()* matches  
63722 the number of digits that will actually be present in the input stream. Historic implementations  
63723 of the %Y conversion specification (with no flags and no minimum field width) produced  
63724 different output formats. Some always produced at least four digits (with 0 fill for years from 0  
63725 through 999) while others only produced the number of digits present in the year (with no fill  
63726 and no padding). These two forms can be produced with the '0' flag and a minimum field  
63727 width options using the conversions specifications %04Y and %01Y, respectively.

63728 In the past, the C and POSIX standards specified that %F produced an ISO 8601:2004 standard  
63729 date format, but didn't specify which one. For years in the range [0001,9999], POSIX.1-2008  
63730 requires that the output produced match the ISO 8601:2004 standard complete representation  
63731 extended format (YYYY-MM-DD) and for years outside of this range produce output that  
63732 matches the ISO 8601:2004 standard expanded representation extended format  
63733 (<+/-><Underline>Y</Underline>YYYY-MM-DD). To fully meet ISO 8601:2004 standard  
63734 requirements, the producer and consumer must agree on a date format that has a specific  
63735 number of bytes reserved to hold the characters used to represent the years that is sufficiently  
63736 large to hold all values that will be shared. For example, the %+13F conversion specification will  
63737 produce output matching the format "<+/->YYYYYY-MM-DD" (a leading '+' or '-' sign; a six-  
63738 digit, 0-filled year; a '-' ; a two-digit, leading 0-filled month; another '-' ; and the two-digit,  
63739 leading 0-filled day within the month).

63740 Note that if the year being printed is greater than 9999, the resulting string from the unadorned  
63741 %F conversion specifications will not conform to the ISO 8601:2004 standard extended format,  
63742 complete representation for a date and will instead be an extended format, expanded  
63743 representation (presumably without the required agreement between the date's producer and  
63744 consumer).

63745 In the C locale, the E and O modifiers are ignored and the replacement strings for the following  
63746 specifiers are:

- |       |    |  |
|-------|----|--|
| 63747 | %a | The first three characters of %A.        |
| 63748 | %A | One of Sunday, Monday, . . . , Saturday. |

63749	%b	The first three characters of %B.
63750	%B	One of January, February, . . . , December.
63751	%c	Equivalent to %a %b %e %T %Y.
63752	%p	One of AM or PM.
63753	%r	Equivalent to %I:%M:%S %p.
63754	%x	Equivalent to %m/%d/%y.
63755	%X	Equivalent to %T.
63756	%Z	Implementation-defined.

#### 63757 RATIONALE

63758 The %Y conversion specification to *strptime()* was frequently assumed to be a four-digit year, but  
 63759 the ISO C standard does not specify that %Y is restricted to any subset of allowed values from the  
 63760 *tm\_year* field. Similarly, the %C conversion specification was assumed to be a two-digit field and  
 63761 the first part of the output from the %F conversion specification was assumed to be a four-digit  
 63762 field. With *tm\_year* being a signed 32 or more-bit **int** and with many current implementations  
 63763 supporting 64-bit **time\_t** types in one or more programming environments, these assumptions  
 63764 are clearly wrong.

63765 POSIX.1-2008 now allows the format specifications %0xC, %0xF, %0xG, and %0xY (where 'x' is  
 63766 a string of decimal digits used to specify printing and scanning of a string of x decimal digits)  
 63767 with leading zero fill characters. Allowing applications to set the field width enables them to  
 63768 agree on the number of digits to be printed and scanned in the ISO 8601:2004 standard  
 63769 expanded representation of a year (for %F, %G, and %Y) or all but the last two digits of the year  
 63770 (for %C). This is based on a feature in some versions of GNU **libc**'s *strptime()*. The GNU version  
 63771 allows specifying space, zero, or no-fill characters in *strptime()* format strings, but does not allow  
 63772 any flags to be specified in *strptime()* format strings. These implementations also allow these  
 63773 flags to be specified for any numeric field. POSIX.1-2008 only requires the zero fill flag ('0') and  
 63774 only requires that it be recognized when processing %C, %F, %G, and %Y specifications when a  
 63775 minimum field width is also specified. The '0' flag is the only flag needed to produce and scan  
 63776 the ISO 8601:2004 standard year fields using the extended format forms. POSIX.1-2008 also  
 63777 allows applications to specify the same flag and field width specifiers to be used in both  
 63778 *strptime()* and *strptime()* format strings for symmetry. Systems may provide other flag characters  
 63779 and may accept flags in conjunction with conversion specifiers other than %C, %F, %G, and %Y;  
 63780 but portable applications cannot depend on such extensions.

63781 POSIX.1-2008 now also allows the format specifications %+xC, %+xF, %+xG, and %+xY (where  
 63782 'x' is a string of decimal digits used to specify printing and scanning of a string of 'x' decimal  
 63783 digits) with leading zero fill characters and a leading '+' sign character if the year being  
 63784 converted is more than four digits or a minimum field width is specified that allows room for  
 63785 more than four digits for the year. This allows date providers and consumers to agree on a  
 63786 specific number of digits to represent a year as required by the ISO 8601:2004 standard  
 63787 expanded representation formats. The expanded representation formats all require the year to  
 63788 begin with a leading '+' or '-' sign. (All of these specifiers can also provide a leading '-'  
 63789 sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or  
 63790 Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm\_year*  
 63791 to assume values corresponding to years before year 1, but the use of such years provided  
 63792 unspecified results.)

63793 Some earlier version of this standard specified that applications wanting to use *strptime()* to scan  
 63794 dates and times printed by *strptime()* should provide non-digit characters between fields to

**strftime()**

63795 separate years from months and days. It also supported %F to print and scan the ISO 8601:2004  
 63796 standard extended format, complete representation date for years 1 through 9999 (i.e., YYYY-  
 63797 MM-DD). However, many applications were written to print (using *strftime()*) and scan (using  
 63798 *strptime()*) dates written using the basic format complete representation (four-digit years) and  
 63799 truncated representation (two-digit years) specified by the ISO 8601:2004 standard  
 63800 representation of dates and times which do not have any separation characters between fields.  
 63801 The ISO 8601:2004 standard also specifies basic format expanded representation where the  
 63802 creator and consumer of these fields agree beforehand to represent years as leading zero-filled  
 63803 strings of an agreed length of more than four digits to represent a year (again with no separation  
 63804 characters when year, month, and day are all displayed). Applications producing and  
 63805 consuming expanded representations are encouraged to use the '+' flag and an appropriate  
 63806 maximum field width to scan the year including the leading sign. Note that even without the  
 63807 '+' flag, years less than zero may be represented with a leading minus-sign for %F, %G, and %Y  
 63808 conversion specifications. Using negative years results in unspecified behavior.

63809 If a format specification %+xF with the field width x greater than 11 is specified and the width is  
 63810 large enough to display the full year, the output string produced will match the ISO 8601:2004  
 63811 standard subclause 4.1.2.4 expanded representation, extended format date representation for a  
 63812 specific day. (For years in the range [1,99 999], %+12F is sufficient for an agreed five-digit year  
 63813 with a leading sign using the ISO 8601:2004 standard expanded representation, extended format  
 63814 for a specific day "<+/->YYYYY-MM-DD".) Note also that years less than 0 may produce a  
 63815 leading minus-sign ('-') when using %Y or %C whether or not the '0' or '+' flags are used.

63816 The difference between the '0' flag and the '+' flag is whether the leading '+' character will  
 63817 be provided for years >9999 as required for the ISO 8601:2004 standard extended representation  
 63818 format containing a year. For example:

Year	Conversion Specification	<i>strftime()</i> Output	<i>strptime()</i> Scan Back
1970	%Y	1970	1970
1970	%+4Y	1970	1970
27	%Y	27 or 0027	27
270	%Y	270 or 0270	270
270	%+4Y	0270	270
17	%C%y	0017	17
270	%C%y	0270	270
12345	%Y	12345	1234*
12345	%+4Y	+12345	123*
12345	%05%Y	12345	12345
270	%+5Y or %+3C%y	+0270	270
12345	%+5Y or %+3C%y1+12345	1234*	
12345	%06Y or %04C%y	012345	12345
12345	%+6Y or %+4C%y	+12345	12345
123456	%08Y or %06C%y	00123456	123456
123456	%+8Y or %+6C%y	+0123456	123456

63837 In the cases above marked with a \* in the *strptime()* scan back field, the implied or specified  
 63838 number of characters scanned by *strptime()* was less than the number of characters output by  
 63839 *strftime()* using the same format; so the remaining digits of the year were dropped when the  
 63840 output date produced by *strftime()* was scanned back in by *strptime()*.

63841 **FUTURE DIRECTIONS**

63842 None.

63843 **SEE ALSO**63844 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *time()*,  
63845 *tzset()*, *uselocale()*, *utime()*63846 XBD Section 7.3.5 (on page 158), [<time.h>](#)63847 **CHANGE HISTORY**

63848 First released in Issue 3.

63849 **Issue 5**63850 The description of %OV is changed to be consistent with %V and defines Monday as the first day  
63851 of the week.

63852 The description of %Oy is clarified.

63853 **Issue 6**

63854 Extensions beyond the ISO C standard are marked.

63855 The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from  
63856 “Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is  
63857 the last week of the previous year, and the next week is week 1”.63858 The following new requirements on POSIX implementations derive from alignment with the  
63859 Single UNIX Specification:

- 63860
- The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.
  - The modified conversion specifiers are added for consistency with the ISO POSIX-2  
63861 standard *date* utility.

63863 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 63864
- The *strptime()* prototype is updated.
  - The DESCRIPTION is extensively revised.
  - The %z conversion specifier is added.

63867 An example is added.

63868 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

63869 **Issue 7**

63870 Austin Group Interpretation 1003.1-2001 #163 is applied.

63871 The *strptime\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
63872 Set Part 4.

**strlen()**63873 **NAME**63874 `strlen, strlen` — get length of fixed size string63875 **SYNOPSIS**

```
63876 #include <string.h>
63877 size_t strlen(const char *s);
63878 CX size_t strlen(const char *s, size_t maxlen);
```

63879 **DESCRIPTION**

63880 CX For `strlen()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

63883 The `strlen()` function shall compute the number of bytes in the string to which `s` points, not including the terminating NUL character.

63885 CX The `strlen()` function shall compute the smaller of the number of bytes in the array to which `s` points, not including the terminating NUL character, or the value of the `maxlen` argument. The `strlen()` function shall never examine more than `maxlen` bytes of the array pointed to by `s`.

63888 **RETURN VALUE**

63889 The `strlen()` function shall return the length of `s`; no return value shall be reserved to indicate an error.

63891 CX The `strlen()` function shall return an integer containing the smaller of either the length of the string pointed to by `s` or `maxlen`.

63893 **ERRORS**

63894 No errors are defined.

63895 **EXAMPLES**63896 **Getting String Lengths**

63897 The following example sets the maximum length of `key` and `data` by using `strlen()` to get the lengths of those strings.

```
63899 #include <string.h>
63900 ...
63901 struct element {
63902     char *key;
63903     char *data;
63904 };
63905 ..
63906 char *key, *data;
63907 int len;

63908 *keylength = *datalength = 0;
63909 ...
63910 if ((len = strlen(key)) > *keylength)
63911     *keylength = len;
63912 if ((len = strlen(data)) > *datalength)
63913     *datalength = len;
63914 ...
```

63915 **APPLICATION USAGE**

63916 None.

63917 **RATIONALE**

63918 None.

63919 **FUTURE DIRECTIONS**

63920 None.

63921 **SEE ALSO**63922 [wcslen\(\)](#)63923 XBD [<string.h>](#)63924 **CHANGE HISTORY**

63925 First released in Issue 1. Derived from Issue 1 of the SVID.

63926 **Issue 5**63927 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not  
63928 *s* itself as was previously stated.63929 **Issue 7**63930 The *strnlen()* function is added from The Open Group Technical Standard, 2006, Extended API  
63931 Set Part 1.

**strncasecmp()**63932 **NAME**63933 `strncasecmp`, `strncasecmp_l` — case-insensitive string comparisons63934 **SYNOPSIS**63935 `#include <strings.h>`63936 `int strncasecmp(const char *s1, const char *s2, size_t n);`63937 `int strncasecmp_l(const char *s1, const char *s2,`63938 `size_t n, locale_t locale);`63939 **DESCRIPTION**63940 Refer to *strcasecmp()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

63941 **NAME**63942 `strncat` — concatenate a string with part of another63943 **SYNOPSIS**63944 `#include <string.h>`63945 `char *strncat(char *restrict s1, const char *restrict s2, size_t n);`63946 **DESCRIPTION**63947 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
63948 conflict between the requirements described here and the ISO C standard is unintentional. This  
63949 volume of POSIX.1-2008 defers to the ISO C standard.63950 The `strncat()` function shall append not more than *n* bytes (a NUL character and bytes that  
63951 follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by  
63952 *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL  
63953 character is always appended to the result. If copying takes place between objects that overlap,  
63954 the behavior is undefined.63955 **RETURN VALUE**63956 The `strncat()` function shall return *s1*; no return value shall be reserved to indicate an error.63957 **ERRORS**

63958 No errors are defined.

63959 **EXAMPLES**

63960 None.

63961 **APPLICATION USAGE**

63962 None.

63963 **RATIONALE**

63964 None.

63965 **FUTURE DIRECTIONS**

63966 None.

63967 **SEE ALSO**63968 [strcat\(\)](#)63969 XBD [<string.h>](#)63970 **CHANGE HISTORY**

63971 First released in Issue 1. Derived from Issue 1 of the SVID.

63972 **Issue 6**63973 The `strncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

**strncmp()**63974 **NAME**63975 `strncmp` — compare part of two strings63976 **SYNOPSIS**63977 `#include <string.h>`63978 `int strncmp(const char *s1, const char *s2, size_t n);`63979 **DESCRIPTION**63980 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
63981 conflict between the requirements described here and the ISO C standard is unintentional. This  
63982 volume of POSIX.1-2008 defers to the ISO C standard.63983 The `strncmp()` function shall compare not more than *n* bytes (bytes that follow a NUL character  
63984 are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.63985 The sign of a non-zero return value is determined by the sign of the difference between the  
63986 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
63987 being compared.63988 **RETURN VALUE**63989 Upon successful completion, `strncmp()` shall return an integer greater than, equal to, or less than  
63990 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the  
63991 possibly null-terminated array pointed to by *s2* respectively.63992 **ERRORS**

63993 No errors are defined.

63994 **EXAMPLES**

63995 None.

63996 **APPLICATION USAGE**

63997 None.

63998 **RATIONALE**

63999 None.

64000 **FUTURE DIRECTIONS**

64001 None.

64002 **SEE ALSO**64003 `strcmp()`64004 XBD `<string.h>`64005 **CHANGE HISTORY**

64006 First released in Issue 1. Derived from Issue 1 of the SVID.

64007 **Issue 6**

64008 Extensions beyond the ISO C standard are marked.

64009 **NAME**64010 `stpncpy`, `strncpy` — copy fixed length string, returning a pointer to the array end64011 **SYNOPSIS**64012 `#include <string.h>`64013 CX `char *stpncpy(char *restrict s1, const char *restrict s2, size_t n);`  
64014 `char *strncpy(char *restrict s1, const char *restrict s2, size_t n);`64015 **DESCRIPTION**64016 CX For `strncpy()`: The functionality described on this reference page is aligned with the ISO C  
64017 standard. Any conflict between the requirements described here and the ISO C standard is  
64018 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.64019 CX The `stpncpy()` and `strncpy()` functions shall copy not more than *n* bytes (bytes that follow a NUL  
64020 character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.64021 If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be  
64022 appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

64023 If copying takes place between objects that overlap, the behavior is undefined.

64024 **RETURN VALUE**64025 CX If a NUL character is written to the destination, the `stpncpy()` function shall return the address of  
64026 the first such NUL character. Otherwise, it shall return `&s1[n]`.64027 The `strncpy()` function shall return *s1*.

64028 No return values are reserved to indicate an error.

64029 **ERRORS**

64030 No errors are defined.

64031 **EXAMPLES**

64032 None.

64033 **APPLICATION USAGE**64034 Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that  
64035 the *s2* and *s1* arrays do not overlap.64036 Character movement is performed differently in different implementations. Thus, overlapping  
64037 moves may yield surprises.64038 If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not  
64039 null-terminated.64040 **RATIONALE**

64041 None.

64042 **FUTURE DIRECTIONS**

64043 None.

64044 **SEE ALSO**64045 `strcpy()`, `wcsncpy()`64046 XBD `<string.h>`64047 **CHANGE HISTORY**

64048 First released in Issue 1. Derived from Issue 1 of the SVID.

**strncpy()**64049 **Issue 6**

64050 The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

64051 **Issue 7**

64052 The *stpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API  
64053 Set Part 1.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

64054 **NAME**64055 `strndup` — duplicate a specific number of bytes from a string64056 **SYNOPSIS**64057 CX `#include <string.h>`64058 `char *strndup(const char *s, size_t size);`64059 **DESCRIPTION**64060 Refer to *strdup()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**strlen()**64061 **NAME**

64062        strlen — get length of fixed size string

64063 **SYNOPSIS**

64064 CX        #include &lt;string.h&gt;

64065        size\_t strlen(const char \*s, size\_t maxlen);

64066 **DESCRIPTION**64067        Refer to *strlen()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

64068 **NAME**

64069 strpbrk — scan a string for a byte

64070 **SYNOPSIS**

64071 #include &lt;string.h&gt;

64072 char \*strpbrk(const char \*s1, const char \*s2);

64073 **DESCRIPTION**

64074 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 64075 conflict between the requirements described here and the ISO C standard is unintentional. This  
 64076 volume of POSIX.1-2008 defers to the ISO C standard.

64077 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte  
 64078 from the string pointed to by *s2*.

64079 **RETURN VALUE**

64080 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no  
 64081 byte from *s2* occurs in *s1*.

64082 **ERRORS**

64083 No errors are defined.

64084 **EXAMPLES**

64085 None.

64086 **APPLICATION USAGE**

64087 None.

64088 **RATIONALE**

64089 None.

64090 **FUTURE DIRECTIONS**

64091 None.

64092 **SEE ALSO**64093 *strchr()*, *strrchr()*

64094 XBD &lt;string.h&gt;

64095 **CHANGE HISTORY**

64096 First released in Issue 1. Derived from Issue 1 of the SVID.

**strptime()**64097 **NAME**64098 `strptime` — date and time conversion64099 **SYNOPSIS**

```
64100 XSI #include <time.h>
64101 char *strptime(const char *restrict buf, const char *restrict format,
64102 struct tm *restrict tm);
```

64103 **DESCRIPTION**

64104 The `strptime()` function shall convert the character string pointed to by `buf` to values which are  
 64105 stored in the `tm` structure pointed to by `tm`, using the format specified by `format`.

64106 The format is composed of zero or more directives. Each directive is composed of one of the  
 64107 following: one or more white-space characters (as specified by `isspace()`); an ordinary character  
 64108 (neither `' % '` nor a white-space character); or a conversion specification.

64109 Each conversion specification is introduced by the `' % '` character after which the following  
 64110 appear in sequence:

- 64111 • An optional flag, the zero character (`' 0 '`) or the `<plus-sign>` character (`' + '`), which is  
 64112 ignored.
- 64113 • An optional field width. If a field width is specified, it shall be interpreted as a string of  
 64114 decimal digits that will determine the maximum number of bytes converted for the  
 64115 conversion rather than the number of bytes specified below in the description of the  
 64116 conversion specifiers.
- 64117 • An optional `E` or `O` modifier.
- 64118 • A terminating conversion specifier character that indicates the type of conversion to be  
 64119 applied.

64120 The conversions are determined using the `LC_TIME` category of the current locale. The  
 64121 application shall ensure that there is white-space or other non-alphanumeric characters between  
 64122 any two conversion specifications unless all of the adjacent conversion specifications convert a  
 64123 known, fixed number of characters. In the following list, the maximum number of characters  
 64124 scanned (excluding the one matching the next directive) is as follows:

- 64125 • If a maximum field width is specified, then that number
- 64126 • Otherwise, the pattern "`{x}`" indicates that the maximum is `x`
- 64127 • Otherwise, the pattern "`[x,y]`" indicates that the value shall fall within the range given  
 64128 (both bounds being inclusive), and the maximum number of characters scanned shall be  
 64129 the maximum required to represent any value in the range without leading zeros and  
 64130 without a leading `<plus-sign>`

64131 The following conversion specifiers are supported.

64132 The results are unspecified if a modifier is specified with a flag or with a minimum field width,  
 64133 or if a field width is specified for any conversion specifier other than `C`, `E`, or `Y`.

- 64134 a The day of the week, using the locale's weekday names; either the abbreviated or full  
 64135 name may be specified.
- 64136 A Equivalent to `%a`.

64137	b	The month, using the locale's month names; either the abbreviated or full name may be specified.
64138		
64139	B	Equivalent to %b.
64140	c	Replaced by the locale's appropriate date and time representation.
64141	C	All but the last two digits of the year {2}; leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required.
64142		
64143		
64144	d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
64145	D	The date as %m/%d/%y.
64146	e	Equivalent to %d.
64147	h	Equivalent to %b.
64148	H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required.
64149		
64150	I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required.
64151		
64152	j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required.
64153		
64154	m	The month number [01,12]; leading zeros shall be permitted but shall not be required.
64155	M	The minute [00,59]; leading zeros shall be permitted but shall not be required.
64156	n	Any white space.
64157	p	The locale's equivalent of a.m. or p.m.
64158	r	12-hour clock time using the AM/PM notation if <code>t_fmt_ampm</code> is not an empty string in the <code>LC_TIME</code> portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
64159		
64160		
64161	R	The time as %H:%M.
64162	S	The seconds [00,60]; leading zeros shall be permitted but shall not be required.
64163	t	Any white space.
64164	T	The time as %H:%M:%S.
64165	U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
64166		
64167	w	The weekday as a decimal number [0,6], with 0 representing Sunday.
64168	W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
64169		
64170	x	The date, using the locale's date format.
64171	X	The time, using the locale's time format.
64172	y	The last two digits of the year. When <i>format</i> contains neither a C conversion specifier nor a Y conversion specifier, values in the range [69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required. A leading '+' or '-'
64173		
64174		
64175		

**strptime()**

- 64176 character shall be permitted before any leading zeros but shall not be required.
- 64177 **Note:** It is expected that in a future version of this standard the default century inferred  
64178 from a 2-digit year will change. (This would apply to all commands accepting a  
64179 2-digit year as input.)
- 64180 **Y** The full year {4}; leading zeros shall be permitted but shall not be required. A leading  
64181 '+' or '-' character shall be permitted before any leading zeros but shall not be  
64182 required.
- 64183 **%** Replaced by %.

**64184 Modified Conversion Specifiers**

64185 Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate that  
64186 an alternative format or specification should be used rather than the one normally used by the  
64187 unmodified conversion specifier. If the alternative format or specification does not exist in the  
64188 current locale, the behavior shall be as if the unmodified conversion specification were used.

- 64189 **%Ec** The locale's alternative appropriate date and time representation.
- 64190 **%EC** The name of the base year (period) in the locale's alternative representation.
- 64191 **%Ex** The locale's alternative date representation.
- 64192 **%EX** The locale's alternative time representation.
- 64193 **%Ey** The offset from **%EC** (year only) in the locale's alternative representation.
- 64194 **%EY** The full alternative year representation.
- 64195 **%Od** The day of the month using the locale's alternative numeric symbols; leading zeros  
64196 shall be permitted but shall not be required.
- 64197 **%Oe** Equivalent to **%Od**.
- 64198 **%OH** The hour (24-hour clock) using the locale's alternative numeric symbols.
- 64199 **%OI** The hour (12-hour clock) using the locale's alternative numeric symbols.
- 64200 **%Om** The month using the locale's alternative numeric symbols.
- 64201 **%OM** The minutes using the locale's alternative numeric symbols.
- 64202 **%OS** The seconds using the locale's alternative numeric symbols.
- 64203 **%OU** The week number of the year (Sunday as the first day of the week) using the locale's  
64204 alternative numeric symbols.
- 64205 **%Ow** The number of the weekday (Sunday=0) using the locale's alternative numeric  
64206 symbols.
- 64207 **%OW** The week number of the year (Monday as the first day of the week) using the locale's  
64208 alternative numeric symbols.
- 64209 **%Oy** The year (offset from **%C**) using the locale's alternative numeric symbols.

64210 A conversion specification composed of white-space characters is executed by scanning input up  
64211 to the first character that is not white-space (which remains unscanned), or until no more  
64212 characters can be scanned.

64213 A conversion specification that is an ordinary character is executed by scanning the next  
64214 character from the buffer. If the character scanned from the buffer differs from the one  
64215 comprising the directive, the directive fails, and the differing and subsequent characters remain

64216 unscanned.

64217 A series of conversion specifications composed of %n, %t, white-space characters, or any  
64218 combination is executed by scanning up to the first character that is not white space (which  
64219 remains unscanned), or until no more characters can be scanned.

64220 Any other conversion specification is executed by scanning characters until a character matching  
64221 the next directive is scanned, or until no more characters can be scanned. These characters  
64222 except the one matching the next directive, are then compared to the locale values associated  
64223 with the conversion specifier. If a match is found, values for the appropriate **tm** structure  
64224 members are set to values corresponding to the locale information. Case is ignored when  
64225 matching items in *buf* such as month or weekday names. If no match is found, *strptime()* fails  
64226 and no more characters are scanned.

#### 64227 RETURN VALUE

64228 Upon successful completion, *strptime()* shall return a pointer to the character following the last  
64229 character parsed. Otherwise, a null pointer shall be returned.

#### 64230 ERRORS

64231 No errors are defined.

#### 64232 EXAMPLES

##### 64233 Convert a Data-Plus-Time String to Broken-Down Time and Then into Seconds

64234 The following example demonstrates the use of *strptime()* to convert a string into broken-down  
64235 time. The broken-down time is then converted into seconds since the Epoch using *mktime()*.

```
64236 #include <time.h>
64237 ...
64238 struct tm tm;
64239 time_t t;
64240 if (strptime("6 Dec 2004 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
64241     /* Handle error */;
64242 printf("year: %d; month: %d; day: %d;\n",
64243        tm.tm_year, tm.tm_mon, tm.tm_mday);
64244 printf("hour: %d; minute: %d; second: %d\n",
64245        tm.tm_hour, tm.tm_min, tm.tm_sec);
64246 printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);
64247 tm.tm_isdst = -1; /* Not set by strptime(); tells mktime()
64248                  to determine whether daylight saving time
64249                  is in effect */
64250 t = mktime(&tm);
64251 if (t == -1)
64252     /* Handle error */;
64253 printf("seconds since the Epoch: %ld\n", (long) t);"
```

#### 64254 APPLICATION USAGE

64255 Several "equivalent to" formats and the special processing of white-space characters are  
64256 provided in order to ease the use of identical *format* strings for *strptime()* and *strptime()*.

64257 It should be noted that dates constructed by the *strptime()* function with the %Y or %C%y  
64258 conversion specifiers may have values larger than 9999. If the *strptime()* function is used to read  
64259 such values using %C%y or %Y, the year values will be truncated to four digits. Applications

**strptime()**

- 64260 should use `%+w%y` or `%+xY` with *w* and *x* set large enough to contain the full value of any years  
64261 that will be printed or scanned.
- 64262 See also the APPLICATION USAGE section in *strptime()*.
- 64263 It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the  
64264 current contents of the structure or overwrite all contents of the structure. Conforming  
64265 applications should make a single call to *strptime()* with a format and all data needed to  
64266 completely specify the date and time being converted.
- 64267 **RATIONALE**
- 64268 See the RATIONALE section for *strptime()*.
- 64269 **FUTURE DIRECTIONS**
- 64270 None.
- 64271 **SEE ALSO**
- 64272 *fprintf()*, *fscanf()*, *strptime()*, *time()*
- 64273 XBD `<time.h>`
- 64274 **CHANGE HISTORY**
- 64275 First released in Issue 4.
- 64276 **Issue 5**
- 64277 Moved from ENHANCED I18N to BASE.
- 64278 The [ENOSYS] error is removed.
- 64279 The exact meaning of the `%y` and `%0y` specifiers is clarified in the DESCRIPTION.
- 64280 **Issue 6**
- 64281 The Open Group Corrigendum U033/5 is applied. The `%r` specifier description is reworded.
- 64282 The normative text is updated to avoid use of the term “must” for application requirements.
- 64283 The **restrict** keyword is added to the *strptime()* prototype for alignment with the  
64284 ISO/IEC 9899:1999 standard.
- 64285 The Open Group Corrigendum U047/2 is applied.
- 64286 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
64287 specification” for consistency with *strptime()*.
- 64288 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the  
64289 EXAMPLES section.
- 64290 **Issue 7**
- 64291 Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and  
64292 APPLICATION USAGE sections.
- 64293 Austin Group Interpretation 1003.1-2001 #163 is applied.
- 64294 SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression  
64295 that `%Y` is 4-digit years.

64296 **NAME**

64297           strrchr — string scanning operation

64298 **SYNOPSIS**

64299           #include &lt;string.h&gt;

64300           char \*strrchr(const char \*s, int c);

64301 **DESCRIPTION**

64302 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 64303 conflict between the requirements described here and the ISO C standard is unintentional. This  
 64304 volume of POSIX.1-2008 defers to the ISO C standard.

64305       The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string  
 64306 pointed to by *s*. The terminating NUL character is considered to be part of the string.

64307 **RETURN VALUE**

64308       Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does  
 64309 not occur in the string.

64310 **ERRORS**

64311       No errors are defined.

64312 **EXAMPLES**64313           **Finding the Base Name of a File**

64314       The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()*  
 64315 function searches backwards through the name of the file to find the last '/' character in *name*.  
 64316 This pointer (plus one) will point to the base name of the file.

```
64317       #include <string.h>
64318       ...
64319       const char *name;
64320       char *basename;
64321       ...
64322       basename = strrchr(name, '/') + 1;
64323       ...
```

64324 **APPLICATION USAGE**

64325       None.

64326 **RATIONALE**

64327       None.

64328 **FUTURE DIRECTIONS**

64329       None.

64330 **SEE ALSO**64331       [strchr\(\)](#)64332       XBD [<string.h>](#)64333 **CHANGE HISTORY**

64334       First released in Issue 1. Derived from Issue 1 of the SVID.

**strsignal()**64335 **NAME**

64336 strsignal — get name of signal

64337 **SYNOPSIS**

```
64338 CX #include <string.h>
64339 char *strsignal(int signum);
```

64340 **DESCRIPTION**

64341 The *strsignal()* function shall map the signal number in *signum* to an implementation-defined string and shall return a pointer to it. It shall use the same set of messages as the *psignal()* function.

64344 The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strsignal()* or *setlocale()*.

64346 The contents of the message strings returned by *strsignal()* should be determined by the setting of the *LC\_MESSAGES* category in the current locale.

64348 The implementation shall behave as if no function defined in this standard calls *strsignal()*.

64349 Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strsignal()*, then check *errno*.

64351 The *strsignal()* function need not be thread-safe.

64352 **RETURN VALUE**

64353 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise, if *signum* is not a valid signal number, the return value is unspecified.

64355 **ERRORS**

64356 No errors are defined.

64357 **EXAMPLES**

64358 None.

64359 **APPLICATION USAGE**

64360 None.

64361 **RATIONALE**

64362 If *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal()* function returns a pointer to a string containing an unspecified message denoting an unknown signal. POSIX.1-2008 leaves this return value unspecified.

64365 **FUTURE DIRECTIONS**

64366 None.

64367 **SEE ALSO**

64368 *psiginfo()*, *setlocale()*

64369 XBD <string.h>

64370 **CHANGE HISTORY**

64371 First released in Issue 7.

64372 **NAME**

64373 strspn — get length of a substring

64374 **SYNOPSIS**

64375 #include &lt;string.h&gt;

64376 size\_t strspn(const char \*s1, const char \*s2);

64377 **DESCRIPTION**

64378 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 64379 conflict between the requirements described here and the ISO C standard is unintentional. This  
 64380 volume of POSIX.1-2008 defers to the ISO C standard.

64381 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
 64382 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

64383 **RETURN VALUE**

64384 The *strspn()* function shall return the computed length; no return value is reserved to indicate an  
 64385 error.

64386 **ERRORS**

64387 No errors are defined.

64388 **EXAMPLES**

64389 None.

64390 **APPLICATION USAGE**

64391 None.

64392 **RATIONALE**

64393 None.

64394 **FUTURE DIRECTIONS**

64395 None.

64396 **SEE ALSO**64397 [strcspn\(\)](#)64398 XBD [<string.h>](#)64399 **CHANGE HISTORY**

64400 First released in Issue 1. Derived from Issue 1 of the SVID.

64401 **Issue 5**

64402 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not  
 64403 *s* itself as was previously stated.

64404 **Issue 7**

64405 SD5-XSH-ERN-182 is applied.

**strstr()**64406 **NAME**64407            **strstr** — find a substring64408 **SYNOPSIS**

64409            #include &lt;string.h&gt;

64410            char \*strstr(const char \*s1, const char \*s2);

64411 **DESCRIPTION**64412 **CX**            The functionality described on this reference page is aligned with the ISO C standard. Any  
64413 conflict between the requirements described here and the ISO C standard is unintentional. This  
64414 volume of POSIX.1-2008 defers to the ISO C standard.64415            The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the  
64416 sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.64417 **RETURN VALUE**64418            Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer  
64419 if the string is not found.64420            If *s2* points to a string with zero length, the function shall return *s1*.64421 **ERRORS**

64422            No errors are defined.

64423 **EXAMPLES**

64424            None.

64425 **APPLICATION USAGE**

64426            None.

64427 **RATIONALE**

64428            None.

64429 **FUTURE DIRECTIONS**

64430            None.

64431 **SEE ALSO**64432            [strchr\(\)](#)64433            XBD [<string.h>](#)64434 **CHANGE HISTORY**

64435            First released in Issue 3. Included for alignment with the ANSI C standard.

## 64436 NAME

64437 strtod, strtodf, strtold — convert a string to a double-precision number

## 64438 SYNOPSIS

64439 #include &lt;stdlib.h&gt;

64440 double strtod(const char \*restrict *nptr*, char \*\*restrict *endptr*);64441 float strtodf(const char \*restrict *nptr*, char \*\*restrict *endptr*);64442 long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);

## 64443 DESCRIPTION

64444 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 64445 conflict between the requirements described here and the ISO C standard is unintentional. This  
 64446 volume of POSIX.1-2008 defers to the ISO C standard.

64447 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,  
 64448 and **long double** representation, respectively. First, they decompose the input string into three  
 64449 parts:

- 64450 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 64451 2. A subject sequence interpreted as a floating-point constant or representing infinity or  
 64452 NaN
- 64453 3. A final string of one or more unrecognized characters, including the terminating NUL  
 64454 character of the input string

64455 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 64456 the result.

64457 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the  
 64458 following:

- 64459 • A non-empty sequence of decimal digits optionally containing a radix character; then an  
 64460 optional exponent part consisting of the character 'e' or the character 'E', optionally  
 64461 followed by a '+' or '-' character, and then followed by one or more decimal digits
- 64462 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 64463 character; then an optional binary exponent part consisting of the character 'p' or the  
 64464 character 'P', optionally followed by a '+' or '-' character, and then followed by one or  
 64465 more decimal digits
- 64466 • One of INF or INFINITY, ignoring case
- 64467 • One of NAN or NAN(*n-char-sequence<sub>opt</sub>*), ignoring case in the NAN part, where:

64468 *n-char-sequence*:

64469 digit

64470 nondigit

64471 *n-char-sequence* digit64472 *n-char-sequence* nondigit

64473 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 64474 with the first non-white-space character, that is of the expected form. The subject sequence  
 64475 contains no characters if the input string is not of the expected form.

64476 If the subject sequence has the expected form for a floating-point number, the sequence of  
 64477 characters starting with the first digit or the decimal-point character (whichever occurs first)  
 64478 shall be interpreted as a floating constant of the C language, except that the radix character shall  
 64479 be used in place of a period, and that if neither an exponent part nor a radix character appears in

**strtod()**

64480 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal  
 64481 floating-point number, an exponent part of the appropriate type with value zero is assumed to  
 64482 follow the last digit in the string. If the subject sequence begins with a minus-sign, the sequence  
 64483 shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an  
 64484 infinity, if representable in the return type, else as if it were a floating constant that is too large  
 64485 for the range of the return type. A character sequence NAN or NAN(*n-char-sequence<sub>opt</sub>*) shall be  
 64486 interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence  
 64487 part that does not have the expected form; the meaning of the *n-char* sequences is  
 64488 implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*,  
 64489 provided that *endptr* is not a null pointer.

64490 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value  
 64491 resulting from the conversion is correctly rounded.

64492 CX The radix character is defined in the locale of the process (category *LC\_NUMERIC*). In the  
 64493 POSIX locale, or in a locale where the radix character is not defined, the radix character shall  
 64494 default to a <period> ( ' . ' ).

64495 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 64496 accepted.

64497 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 64498 performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not  
 64499 a null pointer.

64500 CX The *strtod()* function shall not change the setting of *errno* if successful.

64501 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 64502 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

**RETURN VALUE**

64504 Upon successful completion, these functions shall return the converted value. If no conversion  
 64505 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

64506 If the correct value is outside the range of representable values, ±HUGE\_VAL, ±HUGE\_VALF, or  
 64507 ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to  
 64508 [ERANGE].

64509 If the correct value would cause an underflow, a value whose magnitude is no greater than the  
 64510 smallest normalized positive number in the return type shall be returned and *errno* set to  
 64511 [ERANGE].

**ERRORS**

64512 These functions shall fail if:

64514 CX [ERANGE] The value to be returned would cause overflow or underflow.

64515 These functions may fail if:

64516 CX [EINVAL] No conversion could be performed.

64517 **EXAMPLES**

64518 None.

64519 **APPLICATION USAGE**

64520 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 64521 result is not exactly representable, the result should be one of the two numbers in the  
 64522 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 64523 extra stipulation that the error should have a correct sign for the current rounding direction.

64524 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 64525 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 64526 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 64527 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 64528 values of *L*, *D*, and *U* satisfy  $L \leq D \leq U$ . The result should be one of the (equal or adjacent)  
 64529 values that would be obtained by correctly rounding *L* and *U* according to the current rounding  
 64530 direction, with the extra stipulation that the error with respect to *D* should have a correct sign  
 64531 for the current rounding direction.

64532 The changes to *strtod()* introduced by the ISO/IEC 9899:1999 standard can alter the behavior of  
 64533 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier  
 64534 versions of this standard. One such example would be:

```

64535 int
64536 what_kind_of_number (char *s)
64537 {
64538     char *endp;
64539     double d;
64540     long l;
64541
64542     d = strtod(s, &endp);
64543     if (s != endp && *endp == '\0')
64544         printf("It's a float with value %g\n", d);
64545     else
64546     {
64547         l = strtol(s, &endp, 0);
64548         if (s != endp && *endp == '\0')
64549             printf("It's an integer with value %ld\n", l);
64550         else
64551             return 1;
64552     }
64553     return 0;
64554 }
```

64554 If the function is called with:

64555 `what_kind_of_number ("0x10")`

64556 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

64557 `It's an integer with value 16`

64558 With the ISO/IEC 9899:1999 standard, the result is:

64559 `It's a float with value 16`

64560 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal  
 64561 notation without requiring that either a decimal point or the binary exponent be present.

**strtod()**64562 **RATIONALE**

64563 None.

64564 **FUTURE DIRECTIONS**

64565 None.

64566 **SEE ALSO**64567 *fscanf()*, *isspace()*, *localeconv()*, *setlocale()*, *strtol()*64568 XBD Chapter 7 (on page 135), `<float.h>`, `<stdlib.h>`64569 **CHANGE HISTORY**

64570 First released in Issue 1. Derived from Issue 1 of the SVID.

64571 **Issue 5**64572 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.64573 **Issue 6**

64574 Extensions beyond the ISO C standard are marked.

64575 The following new requirements on POSIX implementations derive from alignment with the  
64576 Single UNIX Specification:

- 64577
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
64578 added if no conversion could be performed.

64579 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 64580
- The *strtod()* function is updated.
  - The *strtodf()* and *strtold()* functions are added.
  - The DESCRIPTION is extensively revised.

64583 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

64584 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second  
64585 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.64586 **Issue 7**

64587 Austin Group Interpretation 1003.1-2001 #015 is applied.

64588 **NAME**64589 `strtoimax, strtoumax` — convert string to integer type64590 **SYNOPSIS**64591 `#include <inttypes.h>`64592 `intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,`  
64593 `int base);`64594 `uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,`  
64595 `int base);`64596 **DESCRIPTION**64597 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
64598 conflict between the requirements described here and the ISO C standard is unintentional. This  
64599 volume of POSIX.1-2008 defers to the ISO C standard.64600 These functions shall be equivalent to the `strtol()`, `strtoll()`, `strtoul()`, and `strtoull()` functions,  
64601 except that the initial portion of the string shall be converted to `intmax_t` and `uintmax_t`  
64602 representation, respectively.64603 **RETURN VALUE**

64604 These functions shall return the converted value, if any.

64605 If no conversion could be performed, zero shall be returned.

64606 If the correct value is outside the range of representable values, `{INTMAX_MAX}`,  
64607 `{INTMAX_MIN}`, or `{UINTMAX_MAX}` shall be returned (according to the return type and sign  
64608 of the value, if any), and `errno` shall be set to `[ERANGE]`.64609 **ERRORS**

64610 These functions shall fail if:

64611 `[ERANGE]` The value to be returned is not representable.

64612 These functions may fail if:

64613 `[EINVAL]` The value of `base` is not supported.64614 **EXAMPLES**

64615 None.

64616 **APPLICATION USAGE**

64617 None.

64618 **RATIONALE**

64619 None.

64620 **FUTURE DIRECTIONS**

64621 None.

64622 **SEE ALSO**64623 `strtol()`, `strtoul()`64624 XBD `<inttypes.h>`64625 **CHANGE HISTORY**

64626 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**strtok()**64627 **NAME**64628 `strtok, strtok_r` — split string into tokens64629 **SYNOPSIS**

```
64630 #include <string.h>
64631 char *strtok(char *restrict s1, const char *restrict s2);
64632 CX char *strtok_r(char *restrict s, const char *restrict sep,
64633 char **restrict lasts);
```

64634 **DESCRIPTION**

64635 CX For `strtok()`: The functionality described on this reference page is aligned with the ISO C  
 64636 standard. Any conflict between the requirements described here and the ISO C standard is  
 64637 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

64638 A sequence of calls to `strtok()` breaks the string pointed to by `s1` into a sequence of tokens, each  
 64639 of which is delimited by a byte from the string pointed to by `s2`. The first call in the sequence  
 64640 has `s1` as its first argument, and is followed by calls with a null pointer as their first argument.  
 64641 The separator string pointed to by `s2` may be different from call to call.

64642 The first call in the sequence searches the string pointed to by `s1` for the first byte that is *not*  
 64643 contained in the current separator string pointed to by `s2`. If no such byte is found, then there  
 64644 are no tokens in the string pointed to by `s1` and `strtok()` shall return a null pointer. If such a byte  
 64645 is found, it is the start of the first token.

64646 The `strtok()` function then searches from there for a byte that *is* contained in the current separator  
 64647 string. If no such byte is found, the current token extends to the end of the string pointed to by  
 64648 `s1`, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is  
 64649 overwritten by a NUL character, which terminates the current token. The `strtok()` function saves  
 64650 a pointer to the following byte, from which the next search for a token shall start.

64651 Each subsequent call, with a null pointer as the value of the first argument, starts searching from  
 64652 the saved pointer and behaves as described above.

64653 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
 64654 `strtok()`.

64655 CX The `strtok()` function need not be thread-safe.

64656 The `strtok_r()` function considers the null-terminated string `s` as a sequence of zero or more text  
 64657 tokens separated by spans of one or more characters from the separator string `sep`. The  
 64658 argument `lasts` points to a user-provided pointer which points to stored information necessary  
 64659 for `strtok_r()` to continue scanning the same string.

64660 In the first call to `strtok_r()`, `s` points to a null-terminated string, `sep` to a null-terminated string of  
 64661 separator characters, and the value pointed to by `lasts` is ignored. The `strtok_r()` function shall  
 64662 return a pointer to the first character of the first token, write a null character into `s` immediately  
 64663 following the returned token, and update the pointer to which `lasts` points.

64664 In subsequent calls, `s` is a null pointer and `lasts` shall be unchanged from the previous call so that  
 64665 subsequent calls shall move through the string `s`, returning successive tokens until no tokens  
 64666 remain. The separator string `sep` may be different from call to call. When no token remains in `s`, a  
 64667 null pointer shall be returned.

64668 **RETURN VALUE**

64669 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,  
64670 if there is no token, *strtok()* shall return a null pointer.

64671 CX The *strtok\_r()* function shall return a pointer to the token found, or a null pointer when no token  
64672 is found.

64673 **ERRORS**

64674 No errors are defined.

64675 **EXAMPLES**64676 **Searching for Word Separators**

64677 The following example searches for tokens separated by <space> characters.

```
64678 #include <string.h>
64679 ...
64680 char *token;
64681 char line[] = "LINE TO BE SEPARATED";
64682 char *search = " ";

64683 /* Token will point to "LINE". */
64684 token = strtok(line, search);

64685 /* Token will point to "TO". */
64686 token = strtok(NULL, search);
```

64687 **Breaking a Line**

64688 The following example uses *strtok()* to break a line into two character strings separated by any  
64689 combination of <space>, <tab>, or <newline> characters.

```
64690 #include <string.h>
64691 ...
64692 struct element {
64693     char *key;
64694     char *data;
64695 };
64696 ...
64697 char line[LINE_MAX];
64698 char *key, *data;
64699 ...
64700 key = strtok(line, " \n");
64701 data = strtok(NULL, " \n");
64702 ...
```

64703 **APPLICATION USAGE**

64704 The *strtok\_r()* function is thread-safe and stores its state in a user-supplied buffer instead of  
64705 possibly using a static data area that may be overwritten by an unrelated call from another  
64706 thread.

64707 **RATIONALE**

64708 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to  
64709 the last substring between separator strings. This function uses static storage to keep track of  
64710 the current string position between calls. The new function, *strtok\_r()*, takes an additional  
64711 argument, *lasts*, to keep track of the current position in the string.

**strtok()**64712 **FUTURE DIRECTIONS**

64713 None.

64714 **SEE ALSO**64715 XBD [<string.h>](#)64716 **CHANGE HISTORY**

64717 First released in Issue 1. Derived from Issue 1 of the SVID.

64718 **Issue 5**64719 The *strtok\_r()* function is included for alignment with the POSIX Threads Extension.64720 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.64721 **Issue 6**

64722 Extensions beyond the ISO C standard are marked.

64723 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

64724 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

64725 The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

64727 The **restrict** keyword is added to the *strtok()* and *strtok\_r()* prototypes for alignment with the ISO/IEC 9899: 1999 standard.64729 **Issue 7**

64730 Austin Group Interpretation 1003.1-2001 #156 is applied.

64731 SD5-XSH-ERN-235 is applied, correcting an example.

64732 The *strtok\_r()* function is moved from the Thread-Safe Functions option to the Base.

## 64733 NAME

64734 `strtol, strtoll` — convert a string to a long integer

## 64735 SYNOPSIS

64736 `#include <stdlib.h>`

```
64737 long strtol(const char *restrict str, char **restrict endptr, int base);
64738 long long strtoll(const char *restrict str, char **restrict endptr,
64739 int base)
```

## 64740 DESCRIPTION

64741 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 64742 conflict between the requirements described here and the ISO C standard is unintentional. This  
 64743 volume of POSIX.1-2008 defers to the ISO C standard.

64744 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and  
 64745 **long long** representation, respectively. First, they decompose the input string into three parts:

- 64746 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 64747 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 64748 value of *base*
- 64749 3. A final string of one or more unrecognized characters, including the terminating NUL  
 64750 character of the input string.

64751 Then they shall attempt to convert the subject sequence to an integer, and return the result.

64752 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 64753 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 64754 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 64755 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 64756 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 64757 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

64758 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 64759 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 64760 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 64761 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 64762 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
 64763 digits, following the sign if present.

64764 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 64765 with the first non-white-space character that is of the expected form. The subject sequence shall  
 64766 contain no characters if the input string is empty or consists entirely of white-space characters,  
 64767 or if the first non-white-space character is other than a sign or a permissible letter or digit.

64768 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 64769 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
 64770 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
 64771 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
 64772 begins with a minus-sign, the value resulting from the conversion shall be negated. A pointer to  
 64773 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 64774 pointer.

64775 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 64776 accepted.

64777 If the subject sequence is empty or does not have the expected form, no conversion is performed;

**strtol()**

64778 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 64779 pointer.

64780 CX The *strtol()* function shall not change the setting of *errno* if successful.

64781 Since 0, {LONG\_MIN} or {LLONG\_MIN}, and {LONG\_MAX} or {LLONG\_MAX} are returned  
 64782 on error and are also valid returns on success, an application wishing to check for error  
 64783 situations should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

**RETURN VALUE**

64785 Upon successful completion, these functions shall return the converted value, if any. If no  
 64786 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

64787 If the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
 64788 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
 64789 *errno* set to [ERANGE].

**ERRORS**

64790 These functions shall fail if:

64792 [ERANGE] The value to be returned is not representable.

64793 These functions may fail if:

64794 CX [EINVAL] The value of *base* is not supported.

**EXAMPLES**

64796 None.

**APPLICATION USAGE**

64798 None.

**RATIONALE**

64800 None.

**FUTURE DIRECTIONS**

64802 None.

**SEE ALSO**

64804 *fscanf()*, *isalpha()*, *strtod()*

64805 XBD <stdlib.h>

**CHANGE HISTORY**

64807 First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

64809 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

**Issue 6**

64811 Extensions beyond the ISO C standard are marked.

64812 The following new requirements on POSIX implementations derive from alignment with the  
 64813 Single UNIX Specification:

- 64814 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 64815 added if no conversion could be performed.

64816 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 64817 • The *strtol()* prototype is updated.
- 64818 • The *strtoll()* function is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**strtold()***System Interfaces*64819 **NAME**64820            **strtold** — convert a string to a double-precision number64821 **SYNOPSIS**

64822            #include &lt;stdlib.h&gt;

64823            long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);64824 **DESCRIPTION**64825            Refer to *strtod()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***strtoll()**64826 **NAME**64827        **strtoll** — convert a string to a long integer64828 **SYNOPSIS**

64829        #include &lt;stdlib.h&gt;

64830        long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,  
64831                    int *base*);64832 **DESCRIPTION**64833        Refer to *strtol()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**strtoul()**64834 **NAME**64835 `strtoul, strtoull` — convert a string to an unsigned long64836 **SYNOPSIS**64837 `#include <stdlib.h>`64838 `unsigned long strtoul(const char *restrict str,`  
64839 `char **restrict endptr, int base);`64840 `unsigned long long strtoull(const char *restrict str,`  
64841 `char **restrict endptr, int base);`64842 **DESCRIPTION**64843 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
64844 conflict between the requirements described here and the ISO C standard is unintentional. This  
64845 volume of POSIX.1-2008 defers to the ISO C standard.64846 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**  
64847 **long** and **unsigned long long** representation, respectively. First, they decompose the input string  
64848 into three parts:

- 64849
1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
  - 64850 2. A subject sequence interpreted as an integer represented in some radix determined by the  
64851 value of *base*
  - 64852 3. A final string of one or more unrecognized characters, including the terminating NUL  
64853 character of the input string

64854 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
64855 result.64856 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
64857 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
64858 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
64859 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
64860 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
64861 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.64862 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
64863 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
64864 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
64865 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
64866 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
64867 digits, following the sign if present.64868 The subject sequence is defined as the longest initial subsequence of the input string, starting  
64869 with the first non-white-space character that is of the expected form. The subject sequence shall  
64870 contain no characters if the input string is empty or consists entirely of white-space characters,  
64871 or if the first non-white-space character is other than a sign or a permissible letter or digit.64872 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
64873 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
64874 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
64875 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
64876 begins with a minus-sign, the value resulting from the conversion shall be negated. A pointer to  
64877 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
64878 pointer.

- 64879 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
64880 accepted.
- 64881 If the subject sequence is empty or does not have the expected form, no conversion shall be  
64882 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*  
64883 is not a null pointer.
- 64884 CX The *strtoul()* function shall not change the setting of *errno* if successful.
- 64885 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and are also valid returns  
64886 on success, an application wishing to check for error situations should set *errno* to 0, then call  
64887 *strtoul()* or *strtoull()*, then check *errno*.
- 64888 **RETURN VALUE**
- 64889 Upon successful completion, these functions shall return the converted value, if any. If no  
64890 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the  
64891 correct value is outside the range of representable values, {ULONG\_MAX} or {ULLONG\_MAX}  
64892 shall be returned and *errno* set to [ERANGE].
- 64893 **ERRORS**
- 64894 These functions shall fail if:
- 64895 CX [EINVAL] The value of *base* is not supported.
- 64896 [ERANGE] The value to be returned is not representable.
- 64897 These functions may fail if:
- 64898 CX [EINVAL] No conversion could be performed.
- 64899 **EXAMPLES**
- 64900 None.
- 64901 **APPLICATION USAGE**
- 64902 None.
- 64903 **RATIONALE**
- 64904 None.
- 64905 **FUTURE DIRECTIONS**
- 64906 None.
- 64907 **SEE ALSO**
- 64908 *fscanf()*, *isalpha()*, *strtod()*, *strtol()*
- 64909 XBD <stdlib.h>
- 64910 **CHANGE HISTORY**
- 64911 First released in Issue 4. Derived from the ANSI C standard.
- 64912 **Issue 5**
- 64913 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 64914 **Issue 6**
- 64915 Extensions beyond the ISO C standard are marked.
- 64916 The following new requirements on POSIX implementations derive from alignment with the  
64917 Single UNIX Specification:
- 64918
- The [EINVAL] error condition is added for when the value of *base* is not supported.

**strtoul()**

64919 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
64920 added if no conversion could be performed.

64921 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 64922 • The *strtoul()* prototype is updated.
- 64923 • The *strtoull()* function is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***strtoumax()**64924 **NAME**64925        **strtoumax** — convert a string to an integer type64926 **SYNOPSIS**

64927        #include &lt;inttypes.h&gt;

64928        uintmax\_t strtoumax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
64929                            int *base*);64930 **DESCRIPTION**64931        Refer to *strtoimax()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**strxfrm()**64932 **NAME**64933 `strxfrm, strxfrm_l` — string transformation64934 **SYNOPSIS**64935 `#include <string.h>`64936 `size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);`64937 CX `size_t strxfrm_l(char *restrict s1, const char *restrict s2,`  
64938 `size_t n, locale_t locale);`64939 **DESCRIPTION**64940 CX For `strxfrm()`: The functionality described on this reference page is aligned with the ISO C  
64941 standard. Any conflict between the requirements described here and the ISO C standard is  
64942 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.64943 CX The `strxfrm()` and `strxfrm_l()` functions shall transform the string pointed to by `s2` and place the  
64944 resulting string into the array pointed to by `s1`. The transformation is such that if `strcmp()`  
64945 is applied to two transformed strings, it shall return a value greater than, equal to, or less than 0,  
64946 CX corresponding to the result of `strcoll()` or `strcoll_l()`, respectively, applied to the same two  
64947 CX original strings with the same locale. No more than `n` bytes are placed into the resulting array  
64948 pointed to by `s1`, including the terminating NUL character. If `n` is 0, `s1` is permitted to be a null  
64949 pointer. If copying takes place between objects that overlap, the behavior is undefined.64950 CX The `strxfrm()` and `strxfrm_l()` functions shall not change the setting of `errno` if successful.64951 Since no return value is reserved to indicate an error, an application wishing to check for error  
64952 CX situations should set `errno` to 0, then call `strxfrm()` or `strxfrm_l()`, then check `errno`.64953 **RETURN VALUE**64954 CX Upon successful completion, `strxfrm()` and `strxfrm_l()` shall return the length of the  
64955 transformed string (not including the terminating NUL character). If the value returned is `n` or  
64956 more, the contents of the array pointed to by `s1` are unspecified.64957 CX On error, `strxfrm()` and `strxfrm_l()` may set `errno` but no return value is reserved to indicate an  
64958 error.64959 **ERRORS**

64960 These functions may fail if:

64961 CX [EINVAL] The string pointed to by the `s2` argument contains characters outside the  
64962 domain of the collating sequence.64963 The `strxfrm_l()` function may fail if:64964 CX [EINVAL] `locale` is not a valid locale object.64965 **EXAMPLES**

64966 None.

64967 **APPLICATION USAGE**64968 The transformation function is such that two transformed strings can be ordered by `strcmp()` as  
64969 appropriate to collating sequence information in the locale of the process (category  
64970 `LC_COLLATE`).64971 The fact that when `n` is 0 `s1` is permitted to be a null pointer is useful to determine the size of the  
64972 `s1` array prior to making the transformation.

64973 **RATIONALE**

64974 None.

64975 **FUTURE DIRECTIONS**

64976 None.

64977 **SEE ALSO**64978 *strcmp()*, *strcoll()*64979 XBD `<string.h>`64980 **CHANGE HISTORY**

64981 First released in Issue 3. Included for alignment with the ISO C standard.

64982 **Issue 5**64983 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.64984 **Issue 6**

64985 Extensions beyond the ISO C standard are marked.

64986 The following new requirements on POSIX implementations derive from alignment with the  
64987 Single UNIX Specification:

- 64988
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
64989 added if no conversion could be performed.

64990 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.64991 **Issue 7**64992 The *strxfrm\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
64993 Set Part 4.

**swab()**64994 **NAME**64995 `swab` — swap bytes64996 **SYNOPSIS**

```
64997 XSI #include <unistd.h>
64998 void swab(const void *restrict src, void *restrict dest,
64999          ssize_t nbytes);
```

65000 **DESCRIPTION**

65001 The `swab()` function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to  
65002 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, `swab()`  
65003 copies and exchanges *nbytes*-1 bytes and the disposition of the last byte is unspecified. If  
65004 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is  
65005 negative, `swab()` does nothing.

65006 **RETURN VALUE**

65007 None.

65008 **ERRORS**

65009 No errors are defined.

65010 **EXAMPLES**

65011 None.

65012 **APPLICATION USAGE**

65013 None.

65014 **RATIONALE**

65015 None.

65016 **FUTURE DIRECTIONS**

65017 None.

65018 **SEE ALSO**

65019 XBD &lt;unistd.h&gt;

65020 **CHANGE HISTORY**

65021 First released in Issue 1. Derived from Issue 1 of the SVID.

65022 **Issue 6**

65023 The **restrict** keyword is added to the `swab()` prototype for alignment with the  
65024 ISO/IEC 9899:1999 standard.

65025 **NAME**

65026 swprintf — print formatted wide-character output

65027 **SYNOPSIS**

65028 #include &lt;stdio.h&gt;

65029 #include &lt;wchar.h&gt;

65030 int swprintf(wchar\_t \*restrict ws, size\_t n,

65031 const wchar\_t \*restrict format, ...);

65032 **DESCRIPTION**65033 Refer to *fwprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**swscanf()**65034 **NAME**65035 `swscanf` — convert formatted wide-character input65036 **SYNOPSIS**65037 `#include <stdio.h>`65038 `#include <wchar.h>`65039 `int swscanf(const wchar_t *restrict ws,`  
65040 `const wchar_t *restrict format, ...);`65041 **DESCRIPTION**65042 Refer to *fwscanf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

65043 **NAME**65044 `symlink, symlinkat` — make a symbolic link relative to directory file descriptor65045 **SYNOPSIS**65046 `#include <unistd.h>`65047 `int symlink(const char *path1, const char *path2);`65048 `int symlinkat(const char *path1, int fd, const char *path2);`65049 **DESCRIPTION**65050 The `symlink()` function shall create a symbolic link called `path2` that contains the string pointed to  
65051 by `path1` (`path2` is the name of the symbolic link created, `path1` is the string contained in the  
65052 symbolic link).65053 The string pointed to by `path1` shall be treated only as a character string and shall not be  
65054 validated as a pathname.65055 If the `symlink()` function fails for any reason other than [EIO], any file named by `path2` shall be  
65056 unaffected.65057 The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's  
65058 group ID shall be set to the group ID of the parent directory or to the effective group ID of the  
65059 process. Implementations shall provide a way to initialize the symbolic link's group ID to the  
65060 group ID of the parent directory. Implementations may, but need not, provide an  
65061 implementation-defined way to initialize the symbolic link's group ID to the effective group ID  
65062 of the calling process.65063 The values of the file mode bits for the created symbolic link are unspecified. All interfaces  
65064 specified by POSIX.1-2008 shall behave as if the contents of symbolic links can always be read,  
65065 except that the value of the file mode bits returned in the `st_mode` field of the `stat` structure is  
65066 unspecified.65067 Upon successful completion, `symlink()` shall mark for update the last data access, last data  
65068 modification, and last file status change timestamps of the symbolic link. Also, the last data  
65069 modification and last file status change timestamps of the directory that contains the new entry  
65070 shall be marked for update.65071 The `symlinkat()` function shall be equivalent to the `symlink()` function except in the case where  
65072 `path2` specifies a relative path. In this case the symbolic link is created relative to the directory  
65073 associated with the file descriptor `fd` instead of the current working directory. If the file  
65074 descriptor was opened without `O_SEARCH`, the function shall check whether directory searches  
65075 are permitted using the current permissions of the directory underlying the file descriptor. If the  
65076 file descriptor was opened with `O_SEARCH`, the function shall not perform the check.65077 If `symlinkat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working  
65078 directory is used and the behavior shall be identical to a call to `symlink()`.65079 **RETURN VALUE**65080 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
65081 return -1 and set `errno` to indicate the error.65082 **ERRORS**

65083 These functions shall fail if:

65084 [EACCES] Write permission is denied in the directory where the symbolic link is being  
65085 created, or search permission is denied for a component of the path prefix of  
65086 `path2`.

## symlink()

65087	[EEXIST]	The <i>path2</i> argument names an existing file or symbolic link.
65088	[EIO]	An I/O error occurs while reading from or writing to the file system.
65089	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path2</i> argument.
65090		
65091	[ENAMETOOLONG]	
65092		The length of a component of the pathname specified by the <i>path2</i> argument is longer than {NAME_MAX} or the length of the <i>path1</i> argument is longer than {SYMLINK_MAX}.
65093		
65094		
65095	[ENOENT]	A component of <i>path2</i> does not name an existing file or <i>path2</i> is an empty string.
65096		
65097	[ENOSPC]	The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which shall contain the link, or the file system is out of file-allocation resources.
65098		
65099		
65100		
65101		
65102	[ENOTDIR]	A component of the path prefix of <i>path2</i> is not a directory.
65103	[EROFS]	The new symbolic link would reside on a read-only file system.
65104		The <i>symlinkat()</i> function shall fail if:
65105	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
65106		
65107	[EBADF]	The <i>path2</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
65108		
65109		These functions may fail if:
65110	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path2</i> argument.
65111		
65112	[ENAMETOOLONG]	
65113		The length of the <i>path2</i> argument exceeds {PATH_MAX} or pathname resolution of a symbolic link in the <i>path2</i> argument produced an intermediate result with a length that exceeds {PATH_MAX}.
65114		
65115		
65116		The <i>symlinkat()</i> function may fail if:
65117	[ENOTDIR]	The <i>path2</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
65118		

**EXAMPLES**

None.

**APPLICATION USAGE**

Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not exist when the link is created. A symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.

65128 **RATIONALE**

65129 Since POSIX.1-2008 does not require any association of file times with symbolic links, there is no  
 65130 requirement that file times be updated by *symlink()*.

65131 The purpose of the *symlinkat()* function is to create symbolic links in directories other than the  
 65132 current working directory without exposure to race conditions. Any part of the path of a file  
 65133 could be changed in parallel to a call to *symlink()*, resulting in unspecified behavior. By opening  
 65134 a file descriptor for the target directory and using the *symlinkat()* function it can be guaranteed  
 65135 that the created symbolic link is located relative to the desired directory.

65136 **FUTURE DIRECTIONS**

65137 None.

65138 **SEE ALSO**

65139 *fdopendir()*, *fstatat()*, *lchown()*, *link()*, *open()*, *readlink()*, *rename()*, *unlink()*

65140 XBD <[unistd.h](#)>

65141 **CHANGE HISTORY**

65142 First released in Issue 4, Version 2.

65143 **Issue 5**

65144 Moved from X/OPEN UNIX extension to BASE.

65145 **Issue 6**

65146 The following changes were made to align with the IEEE P1003.1a draft standard:

- 65147 • The DESCRIPTION text is updated.
- 65148 • The [ELOOP] optional error condition is added.

65149 **Issue 7**

65150 Austin Group Interpretation 1003.1-2001 #143 is applied.

65151 The *symlinkat()* function is added from The Open Group Technical Standard, 2006, Extended  
 65152 API Set Part 2.

65153 Additions have been made describing how *symlink()* sets the user and group IDs and file mode  
 65154 of the symbolic link, and its effect on timestamps.

65155 Changes are made to allow a directory to be opened for searching.

**sync()**65156 **NAME**

65157 sync — schedule file system updates

65158 **SYNOPSIS**

```
65159 XSI #include <unistd.h>  
65160 void sync(void);
```

65161 **DESCRIPTION**65162 The *sync()* function shall cause all information in memory that updates file systems to be  
65163 scheduled for writing out to all file systems.65164 The writing, although scheduled, is not necessarily complete upon return from *sync()*.65165 **RETURN VALUE**65166 The *sync()* function shall not return a value.65167 **ERRORS**

65168 No errors are defined.

65169 **EXAMPLES**

65170 None.

65171 **APPLICATION USAGE**

65172 None.

65173 **RATIONALE**

65174 None.

65175 **FUTURE DIRECTIONS**

65176 None.

65177 **SEE ALSO**65178 [fsync\(\)](#)65179 XBD [<unistd.h>](#)65180 **CHANGE HISTORY**

65181 First released in Issue 4, Version 2.

65182 **Issue 5**

65183 Moved from X/OPEN UNIX extension to BASE.

65184 **NAME**

65185 sysconf — get configurable system variables

65186 **SYNOPSIS**

65187 #include &lt;unistd.h&gt;

65188 long sysconf(int name);

65189 **DESCRIPTION**

65190 The *sysconf()* function provides a method for the application to determine the current value of a  
 65191 configurable system limit or option (*variable*). The implementation shall support all of the  
 65192 variables listed in the following table and may support others.

65193 The *name* argument represents the system variable to be queried. The following table lists the  
 65194 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,  
 65195 and the symbolic constants defined in <unistd.h> that are the corresponding values used for  
 65196 *name*.

Variable	Value of Name
{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
{AIO_MAX}	_SC_AIO_MAX
{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
{ARG_MAX}	_SC_ARG_MAX
{ATEXIT_MAX}	_SC_ATEXIT_MAX
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{CHILD_MAX}	_SC_CHILD_MAX
Clock ticks/second	_SC_CLK_TCK
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
{IOV_MAX}	_SC_IOV_MAX
{LINE_MAX}	_SC_LINE_MAX
{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
{NGROUPS_MAX}	_SC_NGROUPS_MAX
Initial size of <i>getgrgid_r()</i> and <i>getgrnam_r()</i> data buffers	_SC_GETGR_R_SIZE_MAX
Initial size of <i>getpwuid_r()</i> and <i>getpwnam_r()</i> data buffers	_SC_GETPW_R_SIZE_MAX
{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
{OPEN_MAX}	_SC_OPEN_MAX
_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
_POSIX_BARRIERS	_SC_BARRIERS
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
_POSIX_CPUTIME	_SC_CPUTIME
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL

## sysconf()

	Variable	Value of Name
65232		
65233	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
65234	_POSIX_MEMLOCK	_SC_MEMLOCK
65235	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
65236	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
65237	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
65238	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
65239	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
65240	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
65241	_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
65242	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
65243	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
65244	_POSIX_REGEX	_SC_REGEX
65245	_POSIX_SAVED_IDS	_SC_SAVED_IDS
65246	_POSIX_SEMAPHORES	_SC_SEMAPHORES
65247	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
65248	_POSIX_SHELL	_SC_SHELL
65249	_POSIX_SPAWN	_SC_SPAWN
65250	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
65251	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
65252	_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
65253	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
65254	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
65255	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
65256	_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
65257	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
65258	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
65259	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
65260	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
65261	_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
65262	_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
65263	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
65264	_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
65265	_POSIX_THREADS	_SC_THREADS
65266	_POSIX_TIMEOUTS	_SC_TIMEOUTS
65267	_POSIX_TIMERS	_SC_TIMERS
65268	_POSIX_TRACE	_SC_TRACE
65269	_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
65270	_POSIX_TRACE_EVENT_NAME_MAX	_SC_TRACE_EVENT_NAME_MAX
65271	_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
65272	_POSIX_TRACE_LOG	_SC_TRACE_LOG
65273	_POSIX_TRACE_NAME_MAX	_SC_TRACE_NAME_MAX
65274	_POSIX_TRACE_SYS_MAX	_SC_TRACE_SYS_MAX
65275	_POSIX_TRACE_USER_EVENT_MAX	_SC_TRACE_USER_EVENT_MAX
65276	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
65277	_POSIX_VERSION	_SC_VERSION
65278	_POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
65279	_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
65280	_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
65281	_POSIX_V7_LP64_OFFBIG	_SC_V7_LP64_OFFBIG

	Variable	Value of Name
65282		
65283	OB	
65284	_POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
65285	_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
65286	_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64
65287	_POSIX_V6_LP64_OFFBIG	_SC_V6_LP64_OFFBIG
65288	_POSIX2_C_BIND	_SC_2_C_BIND
65289	_POSIX2_C_DEV	_SC_2_C_DEV
65290	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
65291	_POSIX2_FORT_DEV	_SC_2_FORT_DEV
65292	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
65293	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
65294	_POSIX2_PBS	_SC_2_PBS
65295	_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
65296	_POSIX2_PBS_CHECKPOINT	_SC_2_PBS_CHECKPOINT
65297	_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
65298	_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
65299	_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
65300	_POSIX2_SW_DEV	_SC_2_SW_DEV
65301	_POSIX2_UPE	_SC_2_UPE
65302	_POSIX2_VERSION	_SC_2_VERSION
65303	{PAGE_SIZE}	_SC_PAGE_SIZE
65304	{PAGESIZE}	_SC_PAGESIZE
65305	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
65306	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
65307	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
65308	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
65309	{RE_DUP_MAX}	_SC_RE_DUP_MAX
65310	{RTSIG_MAX}	_SC_RTSIG_MAX
65311	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
65312	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
65313	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
65314	{STREAM_MAX}	_SC_STREAM_MAX
65315	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
65316	{TIMER_MAX}	_SC_TIMER_MAX
65317	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
65318	{TZNAME_MAX}	_SC_TZNAME_MAX
65319	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
65320	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
65321	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
65322	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
65323	_XOPEN_SHM	_SC_XOPEN_SHM
65324	_XOPEN_STREAMS	_SC_XOPEN_STREAMS
65325	_XOPEN_UNIX	_SC_XOPEN_UNIX
65326	_XOPEN_UUCP	_SC_XOPEN_UUCP
65327	_XOPEN_VERSION	_SC_XOPEN_VERSION

**RETURN VALUE**

If *name* is an invalid value, *sysconf()* shall return  $-1$  and set *errno* to indicate the error. If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and the variable has no limit, *sysconf()* shall return  $-1$  without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

Otherwise, *sysconf()* shall return the current variable value on the system. The value returned

**sysconf()**

65333 shall not be more restrictive than the corresponding value described to the application when it  
 65334 was compiled with the implementation's `<limits.h>` or `<unistd.h>`. The value shall not change  
 65335 XSI during the lifetime of the calling process, except that `sysconf(_SC_OPEN_MAX)` may return  
 65336 different values before and after a call to `setrlimit()` which changes the `RLIMIT_NOFILE` soft  
 65337 limit.

65338 If the variable corresponding to *name* is dependent on an unsupported option, the results are  
 65339 unspecified.

**ERRORS**

65340 The `sysconf()` function shall fail if:

65342 [EINVAL] The value of the *name* argument is invalid.

**EXAMPLES**

65343 None.  
 65344

**APPLICATION USAGE**

65345 As `-1` is a permissible return value in a successful situation, an application wishing to check for  
 65346 error situations should set `errno` to `0`, then call `sysconf()`, and, if it returns `-1`, check to see if `errno`  
 65347 is non-zero.  
 65348

65349 Application developers should check whether an option, such as `_POSIX_TRACE`, is supported  
 65350 prior to obtaining and using values for related variables, such as `_POSIX_TRACE_NAME_MAX`.

**RATIONALE**

65351 This functionality was added in response to requirements of application developers and of  
 65352 system vendors who deal with many international system configurations. It is closely related to  
 65353 `pathconf()` and `fpathconf()`.  
 65354

65355 Although a conforming application can run on all systems by never demanding more resources  
 65356 than the minimum values published in this volume of POSIX.1-2008, it is useful for that  
 65357 application to be able to use the actual value for the quantity of a resource available on any  
 65358 given system. To do this, the application makes use of the value of a symbolic constant in  
 65359 `<limits.h>` or `<unistd.h>`.

65360 However, once compiled, the application must still be able to cope if the amount of resource  
 65361 available is increased. To that end, an application may need a means of determining the quantity  
 65362 of a resource, or the presence of an option, at execution time.

65363 Two examples are offered:

65364 1. Applications may wish to act differently on systems with or without job control.  
 65365 Applications vendors who wish to distribute only a single binary package to all instances  
 65366 of a computer architecture would be forced to assume job control is never available if it  
 65367 were to rely solely on the `<unistd.h>` value published in this volume of POSIX.1-2008.

65368 2. International applications vendors occasionally require knowledge of the number of clock  
 65369 ticks per second. Without these facilities, they would be required to either distribute their  
 65370 applications partially in source form or to have 50 Hz and 60 Hz versions for the various  
 65371 countries in which they operate.

65372 It is the knowledge that many applications are actually distributed widely in executable form  
 65373 that leads to this facility. If limited to the most restrictive values in the headers, such applications  
 65374 would have to be prepared to accept the most limited environments offered by the smallest  
 65375 microcomputers. Although this is entirely portable, there was a consensus that they should be  
 65376 able to take advantage of the facilities offered by large systems, without the restrictions  
 65377 associated with source and object distributions.

65378 During the discussions of this feature, it was pointed out that it is almost always possible for an  
 65379 application to discern what a value might be at runtime by suitably testing the various functions  
 65380 themselves. And, in any event, it could always be written to adequately deal with error returns  
 65381 from the various functions. In the end, it was felt that this imposed an unreasonable level of  
 65382 complication and sophistication on the application developer.

65383 This runtime facility is not meant to provide ever-changing values that applications have to  
 65384 check multiple times. The values are seen as changing no more frequently than once per system  
 65385 initialization, such as by a system administrator or operator with an automatic configuration  
 65386 program. This volume of POSIX.1-2008 specifies that they shall not change within the lifetime of  
 65387 the process.

65388 Some values apply to the system overall and others vary at the file system or directory level. The  
 65389 latter are described in *fpathconf()*.

65390 Note that all values returned must be expressible as integers. String values were considered, but  
 65391 the additional flexibility of this approach was rejected due to its added complexity of  
 65392 implementation and use.

65393 Some values, such as {PATH\_MAX}, are sometimes so large that they must not be used to, say,  
 65394 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic  
 65395 constant is not even defined in this case.

65396 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is  
 65397 infinite, returning an error indicating that some other resource limit has been reached is  
 65398 conforming behavior.

#### 65399 FUTURE DIRECTIONS

65400 None.

#### 65401 SEE ALSO

65402 *confstr()*, *fpathconf()*

65403 XBD <limits.h>, <unistd.h>

65404 XCU *getconf*

#### 65405 CHANGE HISTORY

65406 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 65407 Issue 5

65408 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 65409 Threads Extension.

65410 The *XBS\_* variables and name values are added to the table of system variables in the  
 65411 DESCRIPTION. These are all marked EX.

#### 65412 Issue 6

65413 The symbol CLK\_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks  
 65414 per second”.

65415 The symbol {PASS\_MAX} is removed.

65416 The following changes were made to align with the IEEE P1003.1a draft standard:

- 65417 • Table entries are added for the following variables: *\_SC\_REGEX*, *\_SC\_SHELL*,
- 65418 *\_SC\_REGEX\_VERSION*, *\_SC\_SYMLoop\_MAX*.

65419 The following *sysconf()* variables and their associated names are added for alignment with  
 65420 IEEE Std 1003.1d-1999:

## sysconf()

65421            \_POSIX\_ADVISORY\_INFO  
 65422            \_POSIX\_CPUTIME  
 65423            \_POSIX\_SPAWN  
 65424            \_POSIX\_SPORADIC\_SERVER  
 65425            \_POSIX\_THREAD\_CPUTIME  
 65426            \_POSIX\_THREAD\_SPORADIC\_SERVER  
 65427            \_POSIX\_TIMEOUTS

65428            The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- 65429            • A statement expressing the dependency of support for some system variables on
- 65430            implementation options is added.
- 65431            • The following system variables are added:

65432            \_POSIX\_BARRIERS  
 65433            \_POSIX\_CLOCK\_SELECTION  
 65434            \_POSIX\_MONOTONIC\_CLOCK  
 65435            \_POSIX\_READER\_WRITER\_LOCKS  
 65436            \_POSIX\_SPIN\_LOCKS  
 65437            \_POSIX\_TYPED\_MEMORY\_OBJECTS

65438            The following system variables are added for alignment with IEEE Std 1003.2d-1994:

65439            \_POSIX2\_PBS  
 65440            \_POSIX2\_PBS\_ACCOUNTING  
 65441            \_POSIX2\_PBS\_LOCATE  
 65442            \_POSIX2\_PBS\_MESSAGE  
 65443            \_POSIX2\_PBS\_TRACK

65444            The following *sysconf()* variables and their associated names are added for alignment with

65445            IEEE Std 1003.1q-2000:

65446            \_POSIX\_TRACE  
 65447            \_POSIX\_TRACE\_EVENT\_FILTER  
 65448            \_POSIX\_TRACE\_INHERIT  
 65449            \_POSIX\_TRACE\_LOG

65450            The macros associated with the *c89* programming models are marked LEGACY, and new

65451            equivalent macros associated with *c99* are introduced.

65452            IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the

65453            DESCRIPTION to denote that the *\_PC\** and *\_SC\** symbols are now required to be supported. A

65454            corresponding change has been made in the Base Definitions volume of POSIX.1-2008. The

65455            deletion in the second paragraph removes some duplicated text. Additional symbols that were

65456            erroneously omitted from this reference page have been added.

65457            IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the

65458            RETURN VALUE section that the value returned for *sysconf(\_SC\_OPEN\_MAX)* may change if a

65459            call to *setrlimit()* adjusts the RLIMIT\_NOFILE soft limit.

65460            IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the

65461            DESCRIPTION to remove an erroneous entry for *\_POSIX\_SYMLOOP\_MAX*. This corrects an

65462            error in IEEE Std 1003.1-2001/Cor 1-2002.

65463            IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing

65464        \_posix\_file\_locking,        \_posix\_multi\_process,        \_posix2\_c\_version,        and  
65465        \_xopen\_xcu\_version (and their associated \_SC\_\* variables) from the DESCRIPTION and  
65466        APPLICATION USAGE sections.

65467        IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following  
65468        constants (and their associated \_SC\_\* variables) to the DESCRIPTION:

65469                \_posix\_ss\_repl\_max  
65470                \_posix\_trace\_event\_name\_max  
65471                \_posix\_trace\_name\_max  
65472                \_posix\_trace\_sys\_max  
65473                \_posix\_trace\_user\_event\_max

65474        The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables  
65475        are dependent on unsupported options, the results are unspecified.

65476        IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing  
65477        \_REGEX\_VERSION and \_SC\_REGEX\_VERSION.

65478        **Issue 7**

65479        Austin Group Interpretation 1003.1-2001 #160 is applied.

65480        SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum  
65481        size of ...” entries in the table in the DESCRIPTION.

65482        The variables for the supported programming environments are updated to be V7 and the  
65483        LEGACY variables are removed.

65484        The following constants are added:

65485                \_posix\_thread\_robust\_prio\_inherit  
65486                \_posix\_thread\_robust\_prio\_protect

65487        The \_XOPEN\_UUCP variable and its associated \_SC\_XOPEN\_UUCP value is added to the table  
65488        of system variables.

**syslog()***System Interfaces*65489 **NAME**

65490 syslog — log a message

65491 **SYNOPSIS**65492 XSI `#include <syslog.h>`65493 `void syslog(int priority, const char *message, ... /* argument */);`65494 **DESCRIPTION**65495 Refer to *closelog()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

65496 **NAME**

65497 system — issue a command

65498 **SYNOPSIS**

65499 #include &lt;stdlib.h&gt;

65500 int system(const char \*command);

65501 **DESCRIPTION**

65502 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 65503 conflict between the requirements described here and the ISO C standard is unintentional. This  
 65504 volume of POSIX.1-2008 defers to the ISO C standard.

65505 If *command* is a null pointer, the *system()* function shall determine whether the host environment  
 65506 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the  
 65507 string pointed to by *command* to that command processor to be executed in an implementation-  
 65508 defined manner; this might then cause the program calling *system()* to behave in a non-  
 65509 conforming manner or to terminate.

65510 CX The *system()* function shall behave as if a child process were created using *fork()*, and the child  
 65511 process invoked the *sh* utility using *execl()* as follows:

65512 `execl(<shell path>, "sh", "-c", command, (char *)0);`

65513 where <shell path> is an unspecified pathname for the *sh* utility. It is unspecified whether the  
 65514 handlers registered with *pthread\_atfork()* are called as part of the creation of the child process.

65515 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the  
 65516 SIGCHLD signal, while waiting for the command to terminate. If this might cause the  
 65517 application to miss a signal that would have killed it, then the application should examine the  
 65518 return value from *system()* and take whatever action is appropriate to the application if the  
 65519 command terminated due to receipt of a signal.

65520 The *system()* function shall not affect the termination status of any child of the calling processes  
 65521 other than the process or processes it itself creates.

65522 The *system()* function shall not return until the child process has terminated.

65523 The *system()* function need not be thread-safe.

65524 **RETURN VALUE**

65525 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor  
 65526 CX is available, or zero if none is available. The *system()* function shall always return non-zero  
 65527 when *command* is NULL.

65528 CX If *command* is not a null pointer, *system()* shall return the termination status of the command  
 65529 language interpreter in the format specified by *waitpid()*. The termination status shall be as  
 65530 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents  
 65531 the command language interpreter from executing after the child process is created, the return  
 65532 value from *system()* shall be as if the command language interpreter had terminated using  
 65533 *exit(127)* or *\_exit(127)*. If a child process cannot be created, or if the termination status for the  
 65534 command language interpreter cannot be obtained, *system()* shall return  $-1$  and set *errno* to  
 65535 indicate the error.

65536 **ERRORS**

65537 CX The *system()* function may set *errno* values as described by *fork()*.

**system()**

65538 In addition, *system()* may fail if:

65539 CX [ECHILD] The status of the child process created by *system()* is no longer available.

**EXAMPLES**

65541 None.

**APPLICATION USAGE**

65543 If the return value of *system()* is not  $-1$ , its value can be decoded through the use of the macros  
65544 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

65545 Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting  
65546 for the child to terminate, the handling of signals in the executed command is as specified by  
65547 *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG\_DFL when *system()*  
65548 is called, then the child is started with SIGINT handling set to SIG\_DFL.

65549 Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two  
65550 processes reading from the same terminal, for example) when the executed command ignores or  
65551 catches one of the signals. It is also usually the correct action when the user has given a  
65552 command to the application to be executed synchronously (as in the '!' command in many  
65553 interactive applications). In either case, the signal should be delivered only to the child process,  
65554 not to the application itself. There is one situation where ignoring the signals might have less  
65555 than the desired effect. This is when the application uses *system()* to perform some task invisible  
65556 to the user. If the user typed the interrupt character ("C", for example) while *system()* is being  
65557 used in this way, one would expect the application to be killed, but only the executed command  
65558 is killed. Applications that use *system()* in this way should carefully check the return status from  
65559 *system()* to see if the executed command was successful, and should take appropriate action  
65560 when the command fails.

65561 Blocking SIGCHLD while waiting for the child to terminate prevents the application from  
65562 catching the signal and obtaining status from *system()*'s child process before *system()* can get the  
65563 status itself.

65564 The context in which the utility is ultimately executed may differ from that in which *system()*  
65565 was called. For example, file descriptors that have the FD\_CLOEXEC flag set are closed, and the  
65566 process ID and parent process ID are different. Also, if the executed utility changes its  
65567 environment variables or its current working directory, that change is not reflected in the caller's  
65568 context.

65569 There is no defined way for an application to find the specific path for the shell. However,  
65570 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

65571 Using the *system()* function in more than one thread in a process or when the SIGCHLD signal is  
65572 being manipulated by more than one thread in a process may produce unexpected results.

**RATIONALE**

65573 The *system()* function should not be used by programs that have set user (or group) ID  
65574 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used  
65575 instead. This prevents any unforeseen manipulation of the environment of the user that could  
65576 cause execution of commands not anticipated by the calling program.  
65577

65578 There are three levels of specification for the *system()* function. The ISO C standard gives the  
65579 most basic. It requires that the function exists, and defines a way for an application to query  
65580 whether a command language interpreter exists. It says nothing about the command language  
65581 or the environment in which the command is interpreted.

65582 POSIX.1-2008 places additional restrictions on *system()*. It requires that if there is a command  
65583 language interpreter, the environment must be as specified by *fork()* and *exec*. This ensures, for

65584 example, that *close-on-exec* works, that file locks are not inherited, and that the process ID is  
 65585 different. It also specifies the return value from *system()* when the command line can be run,  
 65586 thus giving the application some information about the command's completion status.

65587 Finally, POSIX.1-2008 requires the command to be interpreted as in the shell command language  
 65588 defined in the Shell and Utilities volume of POSIX.1-2008.

65589 Note that, *system(NULL)* is required to return non-zero, indicating that there is a command  
 65590 language interpreter. At first glance, this would seem to conflict with the ISO C standard which  
 65591 allows *system(NULL)* to return zero. There is no conflict, however. A system must have a  
 65592 command language interpreter, and is non-conforming if none is present. It is therefore  
 65593 permissible for the *system()* function on such a system to implement the behavior specified by  
 65594 the ISO C standard as long as it is understood that the implementation does not conform to  
 65595 POSIX.1-2008 if *system(NULL)* returns zero.

65596 It was explicitly decided that when *command* is NULL, *system()* should not be required to check  
 65597 to make sure that the command language interpreter actually exists with the correct mode, that  
 65598 there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically,  
 65599 check for such problems as too many existing child processes, and return zero. However, it  
 65600 would be inappropriate to return zero due to such a (presumably) transient condition. If some  
 65601 condition exists that is not under the control of this application and that would cause any  
 65602 *system()* call to fail, that system has been rendered non-conforming.

65603 Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was  
 65604 interrupted with a signal. This error return was removed, and a requirement that *system()* not  
 65605 return until the child has terminated was added. This means that if a *waitpid()* call in *system()*  
 65606 exits with *errno* set to [EINTR], *system()* must reissue the *waitpid()*. This change was made for  
 65607 two reasons:

- 65608 1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling  
 65609 *wait()*, and that could have the undesirable effect of returning the status of children other  
 65610 than the one started by *system()*.
- 65611 2. While it might require a change in some historical implementations, those  
 65612 implementations already have to be changed because they use *wait()* instead of *waitpid()*.

65613 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a  
 65614 successful *system()* call returns.

65615 To conform to POSIX.1-2008, *system()* must use *waitpid()*, or some similar function, instead of  
 65616 *wait()*.

65617 The following code sample illustrates how *system()* might be implemented on an  
 65618 implementation conforming to POSIX.1-2008.

```
65619 #include <signal.h>
65620 int system(const char *cmd)
65621 {
65622     int stat;
65623     pid_t pid;
65624     struct sigaction sa, savintr, savequit;
65625     sigset_t saveblock;
65626     if (cmd == NULL)
65627         return(1);
65628     sa.sa_handler = SIG_IGN;
65629     sigemptyset(&sa.sa_mask);
65630     sa.sa_flags = 0;
```

**system()**

```

65631     sigemptyset(&saveintr.sa_mask);
65632     sigemptyset(&savequit.sa_mask);
65633     sigaction(SIGINT, &sa, &saveintr);
65634     sigaction(SIGQUIT, &sa, &savequit);
65635     sigaddset(&sa.sa_mask, SIGCHLD);
65636     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
65637     if ((pid = fork()) == 0) {
65638         sigaction(SIGINT, &saveintr, (struct sigaction *)0);
65639         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
65640         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
65641         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
65642         _exit(127);
65643     }
65644     if (pid == -1) {
65645         stat = -1; /* errno comes from fork() */
65646     } else {
65647         while (waitpid(pid, &stat, 0) == -1) {
65648             if (errno != EINTR) {
65649                 stat = -1;
65650                 break;
65651             }
65652         }
65653     }
65654     sigaction(SIGINT, &saveintr, (struct sigaction *)0);
65655     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
65656     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
65657     return(stat);
65658 }

```

65659 Note that, while a particular implementation of *system()* (such as the one above) can assume a  
65660 particular path for the shell, such a path is not necessarily valid on another system. The above  
65661 example is not portable, and is not intended to be.

65662 One reviewer suggested that an implementation of *system()* might want to use an environment  
65663 variable such as *SHELL* to determine which command interpreter to use. The supposed  
65664 implementation would use the default command interpreter if the one specified by the  
65665 environment variable was not available. This would allow a user, when using an application that  
65666 prompts for command lines to be processed using *system()*, to specify a different command  
65667 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not  
65668 follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2008, then  
65669 changing *SHELL* would render *system()* non-conforming. This would affect applications that  
65670 expected the specified behavior from *system()*, and since the Shell and Utilities volume of  
65671 POSIX.1-2008 does not mention that *SHELL* affects *system()*, the application would not know  
65672 that it needed to unset *SHELL*.

65673 **FUTURE DIRECTIONS**

65674 None.

65675 **SEE ALSO**65676 *exec*, *pipe()*, *pthread\_atfork()*, *wait()*65677 XBD *<limits.h>*, *<signal.h>*, *<stdlib.h>*, *<sys/wait.h>*65678 XCU *sh*

65679 **CHANGE HISTORY**

65680 First released in Issue 1. Derived from Issue 1 of the SVID.

65681 **Issue 6**

65682 Extensions beyond the ISO C standard are marked.

65683 **Issue 7**

65684 Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this  
65685 function and treatment of *at\_fork()* handlers.

65686 Austin Group Interpretation 1003.1-2001 #156 is applied.

65687 SD5-XSH-ERN-30 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**tan()**65688 **NAME**

65689 tan, tanf, tanl — tangent function

65690 **SYNOPSIS**

```
65691 #include <math.h>
65692 double tan(double x);
65693 float tanf(float x);
65694 long double tanl(long double x);
```

65695 **DESCRIPTION**

65696 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 65697 conflict between the requirements described here and the ISO C standard is unintentional. This  
 65698 volume of POSIX.1-2008 defers to the ISO C standard.

65699 These functions shall compute the tangent of their argument  $x$ , measured in radians.

65700 An application wishing to check for error situations should set *errno* to zero and call  
 65701 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 65702 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 65703 zero, an error has occurred.

65704 **RETURN VALUE**

65705 Upon successful completion, these functions shall return the tangent of  $x$ .

65706 If the correct value would cause underflow, and is not representable, a range error may occur,  
 65707 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

65708 MX If  $x$  is NaN, a NaN shall be returned.

65709 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

65710 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

65711 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 65712 defined value shall be returned.

65713 If the correct value would cause underflow, and is representable, a range error may occur and  
 65714 the correct value shall be returned.

65715 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*  
 65716 shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$ , respectively, with the same sign  
 65717 as the correct value of the function.

65718 **ERRORS**

65719 These functions shall fail if:

65720 MX **Domain Error** The value of  $x$  is  $\pm\text{Inf}$ .

65721 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 65722 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 65723 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 65724 shall be raised.

65725 XSI **Range Error** The result overflows

65726 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 65727 then *errno* shall be set to [ERANGE]. If the integer expression  
 65728 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 65729 floating-point exception shall be raised.

65730 These functions may fail if:

65731 MX Range Error The result underflows, or the value of  $x$  is subnormal.

65732 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
65733 then *errno* shall be set to [ERANGE]. If the integer expression  
65734 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
65735 floating-point exception shall be raised.

## 65736 EXAMPLES

### 65737 Taking the Tangent of a 45-Degree Angle

```
65738 #include <math.h>
65739 ...
65740 double radians = 45.0 * M_PI / 180;
65741 double result;
65742 ...
65743 result = tan (radians);
```

## 65744 APPLICATION USAGE

65745 There are no known floating-point representations such that for a normal argument,  $\tan(x)$  is  
65746 either overflow or underflow.

65747 These functions may lose accuracy when their argument is near a multiple of  $\pi/2$  or is far from  
65748 0.0.

65749 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
65750 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

## 65751 RATIONALE

65752 None.

## 65753 FUTURE DIRECTIONS

65754 None.

## 65755 SEE ALSO

65756 [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

65757 XBD Section 4.19 (on page 116), [<math.h>](#)

## 65758 CHANGE HISTORY

65759 First released in Issue 1. Derived from Issue 1 of the SVID.

### 65760 Issue 5

65761 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
65762 in previous issues.

### 65763 Issue 6

65764 The [tanf\(\)](#) and [tanl\(\)](#) functions are added for alignment with the ISO/IEC 9899:1999 standard.

65765 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
65766 revised to align with the ISO/IEC 9899:1999 standard.

65767 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
65768 marked.

65769 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last  
65770 paragraph in the RETURN VALUE section.

**tanh()**65771 **NAME**

65772 tanh, tanhf, tanhl — hyperbolic tangent functions

65773 **SYNOPSIS**

```
65774 #include <math.h>
65775 double tanh(double x);
65776 float tanhf(float x);
65777 long double tanhl(long double x);
```

65778 **DESCRIPTION**

65779 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 65780 conflict between the requirements described here and the ISO C standard is unintentional. This  
 65781 volume of POSIX.1-2008 defers to the ISO C standard.

65782 These functions shall compute the hyperbolic tangent of their argument  $x$ .

65783 An application wishing to check for error situations should set *errno* to zero and call  
 65784 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 65785 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 65786 zero, an error has occurred.

65787 **RETURN VALUE**

65788 Upon successful completion, these functions shall return the hyperbolic tangent of  $x$ .

65789 MX If  $x$  is NaN, a NaN shall be returned.

65790 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

65791 If  $x$  is  $\pm\text{Inf}$ ,  $\pm 1$  shall be returned.

65792 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

65793 **ERRORS**

65794 These functions may fail if:

65795 MX Range Error The value of  $x$  is subnormal.

65796 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 65797 then *errno* shall be set to [ERANGE]. If the integer expression  
 65798 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 65799 floating-point exception shall be raised.

65800 **EXAMPLES**

65801 None.

65802 **APPLICATION USAGE**

65803 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 65804 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

65805 **RATIONALE**

65806 None.

65807 **FUTURE DIRECTIONS**

65808 None.

65809 **SEE ALSO**

65810 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

65811 XBD Section 4.19 (on page 116), **<math.h>**

65812 **CHANGE HISTORY**

65813 First released in Issue 1. Derived from Issue 1 of the SVID.

65814 **Issue 5**

65815 The DESCRIPTION is updated to indicate how an application should check for an error. This  
65816 text was previously published in the APPLICATION USAGE section.

65817 **Issue 6**

65818 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

65819 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
65820 revised to align with the ISO/IEC 9899:1999 standard.

65821 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
65822 marked.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**tanl()**65823 **NAME**

65824           tanl — tangent function

65825 **SYNOPSIS**

65826           #include &lt;math.h&gt;

65827           long double tanl(long double x);

65828 **DESCRIPTION**65829           Refer to *tan()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

65830 **NAME**

65831 tcdrain — wait for transmission of output

65832 **SYNOPSIS**

```
65833 #include <termios.h>
65834 int tcdrain(int fildev);
```

65835 **DESCRIPTION**

65836 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is  
 65837 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.

65838 Any attempts to use *tcdrain()* from a process which is a member of a background process group  
 65839 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
 65840 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 65841 shall be allowed to perform the operation, and no signal is sent.

65842 **RETURN VALUE**

65843 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 65844 indicate the error.

65845 **ERRORS**

65846 The *tcdrain()* function shall fail if:

- 65847 [EBADF] The *fildev* argument is not a valid file descriptor.
- 65848 [EINTR] A signal interrupted *tcdrain()*.
- 65849 [ENOTTY] The file associated with *fildev* is not a terminal.

65850 The *tcdrain()* function may fail if:

- 65851 [EIO] The process group of the writing process is orphaned, and the writing process  
 65852 is not ignoring or blocking SIGTTOU.

65853 **EXAMPLES**

65854 None.

65855 **APPLICATION USAGE**

65856 None.

65857 **RATIONALE**

65858 None.

65859 **FUTURE DIRECTIONS**

65860 None.

65861 **SEE ALSO**

65862 [tcflush\(\)](#)  
 65863 XBD Chapter 11 (on page 199), [<termios.h>](#)

65864 **CHANGE HISTORY**

65865 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**tcdrain()**65866 **Issue 6**

65867 The following new requirements on POSIX implementations derive from alignment with the  
65868 Single UNIX Specification:

- 65869 • In the DESCRIPTION, the final paragraph is no longer conditional on  
65870 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 65871 • The [EIO] error is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

65872 **NAME**

65873 tcflow — suspend or restart the transmission or reception of data

65874 **SYNOPSIS**

```
65875 #include <termios.h>
65876 int tcflow(int fildev, int action);
```

65877 **DESCRIPTION**

65878 The *tcflow()* function shall suspend or restart transmission or reception of data on the object  
 65879 referred to by *fildev*, depending on the value of *action*. The *fildev* argument is an open file  
 65880 descriptor associated with a terminal.

- 65881 • If *action* is TCOOFF, output shall be suspended.
- 65882 • If *action* is TCOON, suspended output shall be restarted.
- 65883 • If *action* is TCIOFF and *fildev* refers to a terminal device, the system shall transmit a STOP  
 65884 character, which is intended to cause the terminal device to stop transmitting data to the  
 65885 system. If *fildev* is associated with a pseudo-terminal, the STOP character need not be  
 65886 transmitted.
- 65887 • If *action* is TCION and *fildev* refers to a terminal device, the system shall transmit a START  
 65888 character, which is intended to cause the terminal device to start transmitting data to the  
 65889 system. If *fildev* is associated with a pseudo-terminal, the START character need not be  
 65890 transmitted.

65891 The default on the opening of a terminal file is that neither its input nor its output are  
 65892 suspended.

65893 Attempts to use *tcflow()* from a process which is a member of a background process group on a  
 65894 *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
 65895 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 65896 shall be allowed to perform the operation, and no signal is sent.

65897 **RETURN VALUE**

65898 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 65899 indicate the error.

65900 **ERRORS**

65901 The *tcflow()* function shall fail if:

- 65902 [EBADF] The *fildev* argument is not a valid file descriptor.
- 65903 [EINVAL] The *action* argument is not a supported value.
- 65904 [ENOTTY] The file associated with *fildev* is not a terminal.

65905 The *tcflow()* function may fail if:

- 65906 [EIO] The process group of the writing process is orphaned, and the writing process  
 65907 is not ignoring or blocking SIGTTOU.

**tcfLOW()**65908 **EXAMPLES**

65909 None.

65910 **APPLICATION USAGE**

65911 None.

65912 **RATIONALE**

65913 None.

65914 **FUTURE DIRECTIONS**

65915 None.

65916 **SEE ALSO**65917 [tcsendbreak\(\)](#)65918 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)65919 **CHANGE HISTORY**

65920 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

65921 **Issue 6**65922 The following new requirements on POSIX implementations derive from alignment with the  
65923 Single UNIX Specification:

- 65924
- The [EIO] error is added.

65925 **Issue 7**65926 SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and  
65927 STOP characters.

65928 **NAME**

65929 tcflush — flush non-transmitted output data, non-read input data, or both

65930 **SYNOPSIS**

65931 #include &lt;termios.h&gt;

65932 int tcflush(int *fildev*, int *queue\_selector*);65933 **DESCRIPTION**65934 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*  
65935 (an open file descriptor associated with a terminal) but not transmitted, or data received but not  
65936 read, depending on the value of *queue\_selector*:

- 65937 • If *queue\_selector* is TCIFLUSH, it shall flush data received but not read.
- 65938 • If *queue\_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 65939 • If *queue\_selector* is TCIOFLUSH, it shall flush both data received but not read and data  
65940 written but not transmitted.

65941 Attempts to use *tcflush()* from a process which is a member of a background process group on a  
65942 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU  
65943 signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be  
65944 allowed to perform the operation, and no signal is sent.

65945 **RETURN VALUE**65946 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
65947 indicate the error.65948 **ERRORS**65949 The *tcflush()* function shall fail if:

- 65950 [EBADF] The *fildev* argument is not a valid file descriptor.
- 65951 [EINVAL] The *queue\_selector* argument is not a supported value.
- 65952 [ENOTTY] The file associated with *fildev* is not a terminal.

65953 The *tcflush()* function may fail if:

- 65954 [EIO] The process group of the writing process is orphaned, and the writing process  
65955 is not ignoring or blocking SIGTTOU.

65956 **EXAMPLES**

65957 None.

65958 **APPLICATION USAGE**

65959 None.

65960 **RATIONALE**

65961 None.

65962 **FUTURE DIRECTIONS**

65963 None.

65964 **SEE ALSO**65965 [tcdrain\(\)](#)65966 XBD Chapter 11 (on page 199), [<termios.h>](#)

**tcflush()**65967 **CHANGE HISTORY**

65968 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

65969 **Issue 6**

65970 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE  
65971 sections, references to *tcflow()* are replaced with *tcflush()*.

65972 The following new requirements on POSIX implementations derive from alignment with the  
65973 Single UNIX Specification:

- 65974 • In the DESCRIPTION, the final paragraph is no longer conditional on  
65975 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 65976 • The [EIO] error is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

65977 **NAME**

65978 tcgetattr — get the parameters associated with the terminal

65979 **SYNOPSIS**

65980 #include &lt;termios.h&gt;

65981 int tcgetattr(int *fildev*, struct termios \**termios\_p*);65982 **DESCRIPTION**65983 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev* and store them in the **termios** structure referenced by *termios\_p*. The *fildev* argument is an open file descriptor associated with a terminal.65986 The *termios\_p* argument is a pointer to a **termios** structure.65987 The *tcgetattr()* operation is allowed from any process.65988 If the terminal device supports different input and output baud rates, the baud rates stored in the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).65993 **RETURN VALUE**65994 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.65996 **ERRORS**65997 The *tcgetattr()* function shall fail if:65998 [EBADF] The *fildev* argument is not a valid file descriptor.65999 [ENOTTY] The file associated with *fildev* is not a terminal.66000 **EXAMPLES**

66001 None.

66002 **APPLICATION USAGE**

66003 None.

66004 **RATIONALE**66005 Care must be taken when changing the terminal attributes. Applications should always do a *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the terminal state whenever it is done with the terminal. This is necessary because terminal attributes apply to the underlying port and not to each individual open instance; that is, all processes that have used the terminal see the latest attribute changes.

66011 A program that uses these functions should be written to catch all signals and take other appropriate actions to ensure that when the program terminates, whether planned or not, the terminal device's state is restored to its original state.

66014 Existing practice dealing with error returns when only part of a request can be honored is based on calls to the *ioctl()* function. In historical BSD and System V implementations, the corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if some of the requested changes could not be made. Many existing applications assume this behavior and would no longer work correctly if the return value were changed from zero to -1 in this case.

66020 Note that either specification has a problem. When zero is returned, it implies everything

**tcgetattr()**

66021 succeeded even if some of the changes were not made. When -1 is returned, it implies  
66022 everything failed even though some of the changes were made.

66023 Applications that need all of the requested changes made to work properly should follow  
66024 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

**66025 FUTURE DIRECTIONS**

66026 None.

**66027 SEE ALSO**

66028 [tcsetattr\(\)](#)

66029 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)

**66030 CHANGE HISTORY**

66031 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**66032 Issue 6**

66033 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.  
66034 Previously, the number zero was also allowed but was obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

66035 **NAME**

66036 tcgetpgrp — get the foreground process group ID

66037 **SYNOPSIS**

```
66038 #include <unistd.h>
66039 pid_t tcgetpgrp(int fildes);
```

66040 **DESCRIPTION**

66041 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process  
66042 group associated with the terminal.

66043 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does  
66044 not match the process group ID of any existing process group.

66045 The *tcgetpgrp()* function is allowed from a process that is a member of a background process  
66046 group; however, the information may be subsequently changed by a process that is a member of  
66047 a foreground process group.

66048 **RETURN VALUE**

66049 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the  
66050 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to  
66051 indicate the error.

66052 **ERRORS**

66053 The *tcgetpgrp()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 66054 | [EBADF]  | The <i>fildes</i> argument is not a valid file descriptor.  |
| 66055 | [ENOTTY] | The calling process does not have a controlling terminal, or the file is not the<br>66056 controlling terminal. |

66057 **EXAMPLES**

66058 None.

66059 **APPLICATION USAGE**

66060 None.

66061 **RATIONALE**

66062 None.

66063 **FUTURE DIRECTIONS**

66064 None.

66065 **SEE ALSO**

66066 *setsid()*, *setpgid()*, *tcsetpgrp()*  
66067 XBD [<sys/types.h>](#), [<unistd.h>](#)

66068 **CHANGE HISTORY**

66069 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66070 **Issue 6**

66071 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

66072 The following new requirements on POSIX implementations derive from alignment with the  
66073 Single UNIX Specification:

- 66074 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
66075 required for conforming implementations of previous POSIX specifications, it was not  
66076 required for UNIX applications.

## tcgetpgrp()

66077  
66078

- In the DESCRIPTION, text previously conditional on support for `_POSIX_JOB_CONTROL` is now mandatory. This is a FIPS requirement.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

66079 **NAME**

66080 tcgetsid — get the process group ID for the session leader for the controlling terminal

66081 **SYNOPSIS**

66082 #include &lt;termios.h&gt;

66083 pid\_t tcgetsid(int *fildev*);66084 **DESCRIPTION**66085 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal  
66086 specified by *fildev* is the controlling terminal.66087 **RETURN VALUE**66088 Upon successful completion, *tcgetsid()* shall return the process group ID of the session  
66089 associated with the terminal. Otherwise, a value of (**pid\_t**)-1 shall be returned and *errno* set to  
66090 indicate the error.66091 **ERRORS**66092 The *tcgetsid()* function shall fail if:66093 [EBADF] The *fildev* argument is not a valid file descriptor.66094 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
66095 controlling terminal.66096 **EXAMPLES**

66097 None.

66098 **APPLICATION USAGE**

66099 None.

66100 **RATIONALE**

66101 None.

66102 **FUTURE DIRECTIONS**

66103 None.

66104 **SEE ALSO**

66105 XBD &lt;termios.h&gt;

66106 **CHANGE HISTORY**

66107 First released in Issue 4, Version 2.

66108 **Issue 5**

66109 Moved from X/OPEN UNIX extension to BASE.

66110 The [EACCES] error has been removed from the list of mandatory errors, and the description of  
66111 [ENOTTY] has been reworded.66112 **Issue 7**

66113 SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section.

66114 The *tcgetsid()* function is moved from the XSI option to the Base.

**tcsendbreak()**66115 **NAME**66116 `tcsendbreak` — send a break for a specific duration66117 **SYNOPSIS**66118 `#include <termios.h>`66119 `int tcsendbreak(int fildev, int duration);`66120 **DESCRIPTION**

66121 If the terminal is using asynchronous serial data transmission, `tcsendbreak()` shall cause  
 66122 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it  
 66123 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5  
 66124 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period  
 66125 of time.

66126 The *fildev* argument is an open file descriptor associated with a terminal.

66127 If the terminal is not using asynchronous serial data transmission, it is implementation-defined  
 66128 whether `tcsendbreak()` sends data to generate a break condition or returns without taking any  
 66129 action.

66130 Attempts to use `tcsendbreak()` from a process which is a member of a background process group  
 66131 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
 66132 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 66133 shall be allowed to perform the operation, and no signal is sent.

66134 **RETURN VALUE**

66135 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 66136 indicate the error.

66137 **ERRORS**

66138 The `tcsendbreak()` function shall fail if:

66139 [EBADF] The *fildev* argument is not a valid file descriptor.

66140 [ENOTTY] The file associated with *fildev* is not a terminal.

66141 The `tcsendbreak()` function may fail if:

66142 [EIO] The process group of the writing process is orphaned, and the writing process  
 66143 is not ignoring or blocking SIGTTOU.

66144 **EXAMPLES**

66145 None.

66146 **APPLICATION USAGE**

66147 None.

66148 **RATIONALE**

66149 None.

66150 **FUTURE DIRECTIONS**

66151 None.

66152 **SEE ALSO**

66153 XBD [Chapter 11](#) (on page 199), `<termios.h>`

66154 **CHANGE HISTORY**

66155 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66156 **Issue 6**

66157 The following new requirements on POSIX implementations derive from alignment with the  
66158 Single UNIX Specification:

- 66159 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
66160 mandated. This is a FIPS requirement.
- 66161 • The [EIO] error is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**tcsetattr()**66162 **NAME**

66163 tcsetattr — set the parameters associated with the terminal

66164 **SYNOPSIS**

```
66165 #include <termios.h>
66166 int tcsetattr(int fildev, int optional_actions,
66167               const struct termios *termios_p);
```

66168 **DESCRIPTION**

66169 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the  
 66170 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**  
 66171 structure referenced by *termios\_p* as follows:

- 66172 • If *optional\_actions* is TCSANOW, the change shall occur immediately.
- 66173 • If *optional\_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is  
 66174 transmitted. This function should be used when changing parameters that affect output.
- 66175 • If *optional\_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is  
 66176 transmitted, and all input so far received but not read shall be discarded before the change  
 66177 is made.

66178 If the output baud rate stored in the **termios** structure pointed to by *termios\_p* is the zero baud  
 66179 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the  
 66180 line.

66181 If the input baud rate stored in the **termios** structure pointed to by *termios\_p* is 0, the input baud  
 66182 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

66183 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested  
 66184 actions, even if some of the requested actions could not be performed. It shall set all the  
 66185 attributes that the implementation supports as requested and leave all the attributes not  
 66186 supported by the implementation unchanged. If no part of the request can be honored, it shall  
 66187 return  $-1$  and set *errno* to [EINVAL]. If the input and output baud rates differ and are a  
 66188 combination that is not supported, neither baud rate shall be changed. A subsequent call to  
 66189 *tcgetattr()* shall return the actual state of the terminal device (reflecting both the changes made  
 66190 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values  
 66191 found in the **termios** structure under any circumstances.

66192 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios\_p*  
 66193 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify  
 66194 only fields and flags defined by this volume of POSIX.1-2008 between the call to *tcgetattr()* and  
 66195 *tcsetattr()*, leaving all other fields and flags unmodified.

66196 No actions defined by this volume of POSIX.1-2008, other than a call to *tcsetattr()*, a close of the  
 66197 last file descriptor in the system associated with this terminal device, or an open of the first file  
 66198 descriptor in the system associated with this terminal device (using the O\_TTY\_INIT flag if it is  
 66199 non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes  
 66200 defined by this volume of POSIX.1-2008 to change.

66201 If *tcsetattr()* is called from a process which is a member of a background process group on a *fildev*  
 66202 associated with its controlling terminal:

- 66203 • If the calling process is blocking or ignoring SIGTTOU signals, the operation completes  
 66204 normally and no signal is sent.

- 66205 • Otherwise, a SIGTTOU signal shall be sent to the process group.

**66206 RETURN VALUE**

66207 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
66208 indicate the error.

**66209 ERRORS**

66210 The *tcsetattr()* function shall fail if:

66211 [EBADF] The *fildev* argument is not a valid file descriptor.

66212 [EINTR] A signal interrupted *tcsetattr()*.

66213 [EINVAL] The *optional\_actions* argument is not a supported value, or an attempt was  
66214 made to change an attribute represented in the **termios** structure to an  
66215 unsupported value.

66216 [ENOTTY] The file associated with *fildev* is not a terminal.

66217 The *tcsetattr()* function may fail if:

66218 [EIO] The process group of the writing process is orphaned, and the writing process  
66219 is not ignoring or blocking SIGTTOU.

**66220 EXAMPLES**

66221 None.

**66222 APPLICATION USAGE**

66223 If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to  
66224 determine what baud rates were actually selected.

66225 In general, there are two reasons for an application to change the parameters associated with a  
66226 terminal device:

66227 1. The device already has working parameter settings but the application needs a different  
66228 behavior, such as non-canonical mode instead of canonical mode. The application sets (or  
66229 clears) only a few flags or *c\_cc[]* values. Typically, the terminal device in this case is either  
66230 the controlling terminal for the process or a pseudo-terminal.

66231 2. The device is a modem or similar piece of equipment connected by a serial line, and it  
66232 was not open before the application opened it. In this case, the application needs to  
66233 initialize all of the parameter settings "from scratch". However, since the **termios**  
66234 structure may include both standard and non-standard parameters, the application  
66235 cannot just initialize the whole structure in an arbitrary way (e.g., using *memset()*) as this  
66236 may cause some of the non-standard parameters to be set incorrectly, resulting in non-  
66237 conforming behavior of the terminal device. Conversely, the application cannot just set  
66238 the standard parameters, assuming that the non-standard parameters will already have  
66239 suitable values, as the device might previously have been used with non-conforming  
66240 parameter settings (and some implementations retain the settings from one use to the  
66241 next). The solution is to open the terminal device using the O\_TTY\_INIT flag to initialize  
66242 the terminal device to have conforming parameter settings, obtain those settings using  
66243 *tcgetattr()*, and then set all of the standard parameters to the desired settings.

**66244 RATIONALE**

66245 The *tcsetattr()* function can be interrupted in the following situations:

- 66246 • It is interrupted while waiting for output to drain.

**tcsetattr()**

66247 • It is called from a process in a background process group and SIGTTOU is caught.

66248 See also the RATIONALE section in *tcgetattr()*.

**FUTURE DIRECTIONS**

66250 Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be supported in a future version of this volume of POSIX.1-2008.

**SEE ALSO**

66253 *cfgetispeed()*, *tcgetattr()*

66254 XBD Chapter 11 (on page 199), `<termios.h>`

**CHANGE HISTORY**

66256 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**Issue 6**

66257 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

66260 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

66262 • The [EIO] error is added.

66263 In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of a background process group is clarified.

**Issue 7**

66266 Austin Group Interpretation 1003.1-2001 #144 is applied.

66267 **NAME**

66268 tcsetpgrp — set the foreground process group ID

66269 **SYNOPSIS**

66270 #include &lt;unistd.h&gt;

66271 int tcsetpgrp(int *fildev*, pid\_t *pgid\_id*);66272 **DESCRIPTION**

66273 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID  
 66274 associated with the terminal to *pgid\_id*. The application shall ensure that the file associated with  
 66275 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently  
 66276 associated with the session of the calling process. The application shall ensure that the value of  
 66277 *pgid\_id* matches a process group ID of a process in the same session as the calling process.

66278 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on  
 66279 a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
 66280 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 66281 shall be allowed to perform the operation, and no signal is sent.

66282 **RETURN VALUE**

66283 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 66284 indicate the error.

66285 **ERRORS**66286 The *tcsetpgrp()* function shall fail if:66287 [EBADF] The *fildev* argument is not a valid file descriptor.66288 [EINVAL] This implementation does not support the value in the *pgid\_id* argument.

66289 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
 66290 controlling terminal, or the controlling terminal is no longer associated with  
 66291 the session of the calling process.

66292 [EPERM] The value of *pgid\_id* is a value supported by the implementation, but does not  
 66293 match the process group ID of a process in the same session as the calling  
 66294 process.

66295 **EXAMPLES**

66296 None.

66297 **APPLICATION USAGE**

66298 None.

66299 **RATIONALE**

66300 None.

66301 **FUTURE DIRECTIONS**

66302 None.

66303 **SEE ALSO**66304 *tcgetpgrp()*

66305 XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

66306 **CHANGE HISTORY**

66307 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**tcsetpgrp()**66308 **Issue 6**

66309 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

66310 The following new requirements on POSIX implementations derive from alignment with the  
66311 Single UNIX Specification:

66312 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
66313 required for conforming implementations of previous POSIX specifications, it was not  
66314 required for UNIX applications.

66315 • In the DESCRIPTION and ERRORS sections, text previously conditional on  
66316 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

66317 The normative text is updated to avoid use of the term “must” for application requirements.

66318 The Open Group Corrigendum U047/4 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

66319 **NAME**

66320 tdelete, tfind, tsearch, twalk — manage a binary search tree

66321 **SYNOPSIS**

```

66322 XSI #include <search.h>
66323 void *tdelete(const void *restrict key, void **restrict rootp,
66324             int(*compar)(const void *, const void *));
66325 void *tfind(const void *key, void *const *rootp,
66326            int(*compar)(const void *, const void *));
66327 void *tsearch(const void *key, void **rootp,
66328              int (*compar)(const void *, const void *));
66329 void twalk(const void *root,
66330            void (*action)(const void *, VISIT, int));

```

66331 **DESCRIPTION**

66332 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.  
66333 Comparisons are made with a user-supplied routine, the address of which is passed as the  
66334 *compar* argument. This routine is called with two arguments, which are the pointers to the  
66335 elements being compared. The application shall ensure that the user-supplied routine returns an  
66336 integer less than, equal to, or greater than 0, according to whether the first argument is to be  
66337 considered less than, equal to, or greater than the second argument. The comparison function  
66338 need not compare every byte, so arbitrary data may be contained in the elements in addition to  
66339 the values being compared.

66340 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element  
66341 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed  
66342 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key*  
66343 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a  
66344 pointer to this node returned. Only pointers are copied, so the application shall ensure that the  
66345 calling routine stores the data. The *rootp* argument points to a variable that points to the root  
66346 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;  
66347 in this case, the variable shall be set to point to the node which shall be at the root of the new  
66348 tree.

66349 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.  
66350 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the  
66351 same as for *tsearch()*.

66352 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same  
66353 as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the  
66354 root of the tree. The *tdelete()* function shall return a pointer to the parent of the deleted node, or  
66355 an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the  
66356 node is not found.

66357 If *tsearch()* adds an element to a tree, or *tdelete()* successfully deletes an element from a tree, the  
66358 concurrent use of that tree in another thread, or use of pointers produced by a previous call to  
66359 *tfind()* or *tsearch()*, produces undefined results.

66360 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root  
66361 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below  
66362 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine  
66363 is, in turn, called with three arguments. The first argument shall be the address of the node being  
66364 visited. The structure pointed to by this argument is unspecified and shall not be modified by  
66365 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-

**tdelete()**

66366 element to access the element stored in the node. The second argument shall be a value from an  
66367 enumeration data type:

```
66368 typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

66369 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the  
66370 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a  
66371 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

66372 If the calling function alters the pointer to the root, the result is undefined.

66373 If the functions pointed to by *action* or *compar* (for any of these binary search functions) change  
66374 the tree, the results are undefined.

66375 These functions are thread-safe only as long as multiple threads do not access the same tree.

**RETURN VALUE**

66376 If the node is found, both *tsearch()* and *tfind()* shall return a pointer to it. If not, *tfind()* shall  
66377 return a null pointer, and *tsearch()* shall return a pointer to the inserted item.

66379 A null pointer shall be returned by *tsearch()* if there is not enough space available to create a new  
66380 node.

66381 A null pointer shall be returned by *tdelete()*, *tfind()*, and *tsearch()* if *rootp* is a null pointer on  
66382 entry.

66383 The *tdelete()* function shall return a pointer to the parent of the deleted node, or an unspecified  
66384 non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

66385 The *twalk()* function shall not return a value.

**ERRORS**

66386 No errors are defined.  
66387

**EXAMPLES**

66388 The following code reads in strings and stores structures containing a pointer to each string and  
66389 a count of its length. It then walks the tree, printing out the stored strings and their lengths in  
66390 alphabetical order.  
66391

```
66392 #include <search.h>
66393 #include <string.h>
66394 #include <stdio.h>
66395 #define STRSZ 10000
66396 #define NODSZ 500
66397 struct node { /* Pointers to these are stored in the tree. */
66398     char *string;
66399     int length;
66400 };
66401 char string_space[STRSZ]; /* Space to store strings. */
66402 struct node nodes[NODSZ]; /* Nodes to store. */
66403 void *root = NULL; /* This points to the root. */
66404 int main(int argc, char *argv[])
66405 {
66406     char *strptr = string_space;
66407     struct node *nodeptr = nodes;
66408     void print_node(const void *, VISIT, int);
```

```

66409         int    i = 0, node_compare(const void *, const void *);
66410         while (gets(strptr) != NULL && i++ < NODSZ) {
66411             /* Set node. */
66412             nodeptr->string = strptr;
66413             nodeptr->length = strlen(strptr);
66414             /* Put node into the tree. */
66415             (void) tsearch((void *)nodeptr, (void **)&root,
66416                 node_compare);
66417             /* Adjust pointers, so we do not overwrite tree. */
66418             strptr += nodeptr->length + 1;
66419             nodeptr++;
66420         }
66421         twalk(root, print_node);
66422         return 0;
66423     }
66424     /*
66425     * This routine compares two nodes, based on an
66426     * alphabetical ordering of the string field.
66427     */
66428     int
66429     node_compare(const void *node1, const void *node2)
66430     {
66431         return strcmp(((const struct node *) node1)->string,
66432             ((const struct node *) node2)->string);
66433     }
66434     /*
66435     * This routine prints out a node, the second time
66436     * twalk encounters it or if it is a leaf.
66437     */
66438     void
66439     print_node(const void *ptr, VISIT order, int level)
66440     {
66441         const struct node *p = *(const struct node **) ptr;
66442         if (order == postorder || order == leaf) {
66443             (void) printf("string = %s, length = %d\n",
66444                 p->string, p->length);
66445         }
66446     }

```

**APPLICATION USAGE**

The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()* and *tsearch()*.

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which could result in some confusion over the meaning of **postorder**.

Since the return value of *tdelete()* is an unspecified non-null pointer in the case that the root of the tree has been deleted, applications should only use the return value of *tdelete()* as indication

**tdelete()**

66457 of success or failure and should not assume it can be dereferenced. Some implementations in this  
66458 case will return a pointer to the new root of the tree (or to an empty tree if the deleted root node  
66459 was the only node in the tree); other implementations return arbitrary non-null pointers.

**RATIONALE**

66460 None.  
66461

**FUTURE DIRECTIONS**

66462 None.  
66463

**SEE ALSO**

66464 *hcreate()*, *lsearch()*

66465 XBD <[search.h](#)>

**CHANGE HISTORY**

66467 First released in Issue 1. Derived from Issue 1 of the SVID.  
66468

**Issue 5**

66469 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
66470 previous issues.  
66471

**Issue 6**

66472 The normative text is updated to avoid use of the term “must” for application requirements.  
66473

66474 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the  
66475 ISO/IEC 9899:1999 standard.

**Issue 7**

66476 Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in  
66477 another thread.  
66478

66479 Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete()* when  
66480 the deleted node is the root node.

66481 Austin Group Interpretation 1003.1-2001 #153 is applied.

66482 **NAME**

66483 telldir — current location of a named directory stream

66484 **SYNOPSIS**

```
66485 XSI #include <dirent.h>
66486 long telldir(DIR *dirp);
```

66487 **DESCRIPTION**

66488 The *telldir()* function shall obtain the current location associated with the directory stream  
 66489 specified by *dirp*.

66490 If the most recent operation on the directory stream was a *seekdir()*, the directory position  
 66491 returned from the *telldir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.

66492 **RETURN VALUE**

66493 Upon successful completion, *telldir()* shall return the current location of the specified directory  
 66494 stream.

66495 **ERRORS**

66496 No errors are defined.

66497 **EXAMPLES**

66498 None.

66499 **APPLICATION USAGE**

66500 None.

66501 **RATIONALE**

66502 None.

66503 **FUTURE DIRECTIONS**

66504 None.

66505 **SEE ALSO**

66506 *fdopendir()*, *readdir()*, *seekdir()*

66507 XBD [<dirent.h>](#)

66508 **CHANGE HISTORY**

66509 First released in Issue 2.

**tempnam()**66510 **NAME**

66511 tempnam — create a name for a temporary file

66512 **SYNOPSIS**

```
66513 OB XSI #include <stdio.h>
66514 char *tempnam(const char *dir, const char *pfx);
```

66515 **DESCRIPTION**66516 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

66517 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument  
 66518 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or  
 66519 points to a string which is not a name for an appropriate directory, the path prefix defined as  
 66520 P\_tmpdir in the **<stdio.h>** header shall be used. If that directory is not accessible, an  
 66521 implementation-defined directory may be used.

66522 Many applications prefer their temporary files to have certain initial letter sequences in their  
 66523 names. The *pfx* argument should be used for this. This argument may be a null pointer or point  
 66524 to a string of up to five bytes to be used as the beginning of the filename.

66525 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if  
 66526 called more than {TMP\_MAX} times in a single process, the behavior is implementation-defined.

66527 **RETURN VALUE**

66528 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated  
 66529 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a  
 66530 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the  
 66531 error.

66532 **ERRORS**

66533 The *tempnam()* function shall fail if:

66534 [ENOMEM] Insufficient storage space is available.

66535 **EXAMPLES**66536 **Generating a Pathname**

66537 The following example generates a pathname for a temporary file in directory **/tmp**, with the  
 66538 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to  
 66539 store the filename.

```
66540 #include <stdio.h>
66541 #include <stdlib.h>
66542 .
66543 char *directory = "/tmp";
66544 char *fileprefix = "file";
66545 char *file;

66546 file = tempnam(directory, fileprefix);
66547 free(file);
```

66548 **APPLICATION USAGE**

66549 This function only creates pathnames. It is the application's responsibility to create and remove  
 66550 the files. Between the time a pathname is created and the file is opened, it is possible for some  
 66551 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

66552 Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead of the

66553 obsolescent *tempnam()* function.

66554 **RATIONALE**

66555 None.

66556 **FUTURE DIRECTIONS**

66557 The *tempnam()* function may be removed in a future version.

66558 **SEE ALSO**

66559 *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*

66560 XBD <stdio.h>

66561 **CHANGE HISTORY**

66562 First released in Issue 1. Derived from Issue 1 of the SVID.

66563 **Issue 5**

66564 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
66565 previous issues.

66566 **Issue 7**

66567 The *tempnam()* function is marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**tfind()**66568 **NAME**

66569           tfind — search binary search tree

66570 **SYNOPSIS**

```
66571 XSI       #include <search.h>
66572       void *tfind(const void *key, void *const *rootp,
66573           int (*compar)(const void *, const void *));
```

66574 **DESCRIPTION**66575       Refer to *tdelete()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

66576 **NAME**

66577 tgamma, tgammaf, tgammal — compute gamma() function

66578 **SYNOPSIS**

```
66579 #include <math.h>
66580 double tgamma(double x);
66581 float tgammaf(float x);
66582 long double tgammal(long double x);
```

66583 **DESCRIPTION**

66584 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 66585 conflict between the requirements described here and the ISO C standard is unintentional. This  
 66586 volume of POSIX.1-2008 defers to the ISO C standard.

66587 These functions shall compute the *gamma()* function of *x*.

66588 An application wishing to check for error situations should set *errno* to zero and call  
 66589 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 66590 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 66591 zero, an error has occurred.

66592 **RETURN VALUE**

66593 Upon successful completion, these functions shall return *Gamma(x)*.

66594 CX If *x* is a negative integer, a domain error may occur and either a NaN (if supported) or an  
 66595 MX implementation-defined value shall be returned. On systems that support the IEC 60559  
 66596 Floating-Point option, a domain error shall occur and a NaN shall be returned.

66597 If *x* is  $\pm 0$ , *tgamma()*, *tgammaf()*, and *tgammal()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  
 66598 MX  $\pm$ HUGE\_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, a  
 66599 pole error shall occur;

66600 CX otherwise, a pole error may occur.

66601 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,  
 66602 and *tgammal()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, or  $\pm$ HUGE\_VALL, respectively, with  
 66603 the same sign as the correct value of the function.

66604 MX If *x* is NaN, a NaN shall be returned.

66605 If *x* is +Inf, *x* shall be returned.

66606 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 66607 defined value shall be returned.

66608 **ERRORS**

66609 These functions shall fail if:

66610 MX **Domain Error** The value of *x* is a negative integer, or *x* is -Inf.

66611 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 66612 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 66613 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 66614 shall be raised.

66615 MX **Pole Error** The value of *x* is zero.

66616 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 66617 then *errno* shall be set to [ERANGE]. If the integer expression  
 66618 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 66619 floating-point exception shall be raised.

**tgamma()**

66620	Range Error	The value overflows.
66621		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66622		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
66623		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
66624		floating-point exception shall be raised.
66625	These functions may fail if:	
66626	Domain Error	The value of <i>x</i> is a negative integer.
66627		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66628		then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i>
66629		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
66630		shall be raised.
66631	Pole Error	The value of <i>x</i> is zero.
66632		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66633		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
66634		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
66635		floating-point exception shall be raised.

66636 **EXAMPLES**

66637 None.

66638 **APPLICATION USAGE**66639 For IEEE Std 754-1985 **double**, overflow happens when  $0 < x < 1/\text{DBL\_MAX}$ , and  $171.7 < x$ .66640 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
66641 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.66642 **RATIONALE**66643 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and  
66644 *lgamma()* functions.66645 **FUTURE DIRECTIONS**66646 It is possible that the error response for a negative integer argument may be changed to a pole  
66647 error and a return value of  $\pm\text{Inf}$ .66648 **SEE ALSO**66649 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [lgamma\(\)](#)66650 XBD Section 4.19 (on page 116), [<math.h>](#)66651 **CHANGE HISTORY**

66652 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

66653 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third  
66654 paragraph in the RETURN VALUE section.66655 **Issue 7**

66656 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

66657 **NAME**66658 `time` — get time66659 **SYNOPSIS**

```
66660 #include <time.h>
66661 time_t time(time_t *tloc);
```

66662 **DESCRIPTION**

66663 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 66664 conflict between the requirements described here and the ISO C standard is unintentional. This  
 66665 volume of POSIX.1-2008 defers to the ISO C standard.

66666 CX The `time()` function shall return the value of time in seconds since the Epoch.

66667 The `tloc` argument points to an area where the return value is also stored. If `tloc` is a null pointer,  
 66668 no value is stored.

66669 **RETURN VALUE**

66670 Upon successful completion, `time()` shall return the value of time. Otherwise, `(time_t)-1` shall be  
 66671 returned.

66672 **ERRORS**

66673 No errors are defined.

66674 **EXAMPLES**66675 **Getting the Current Time**

66676 The following example uses the `time()` function to calculate the time elapsed, in seconds, since  
 66677 the Epoch, `localtime()` to convert that value to a broken-down time, and `asctime()` to convert the  
 66678 broken-down time values into a printable string.

```
66679 #include <stdio.h>
66680 #include <time.h>
66681 int main(void)
66682 {
66683     time_t result;
66684     result = time(NULL);
66685     printf("%s%ju secs since the Epoch\n",
66686           asctime(localtime(&result)),
66687           (uintmax_t)result);
66688     return(0);
66689 }
```

66690 This example writes the current time to `stdout` in a form like this:

```
66691 Wed Jun 26 10:32:15 1996
66692 835810335 secs since the Epoch
```

**time()**66693 **Timing an Event**

66694 The following example gets the current time, prints it out in the user's format, and prints the  
66695 number of minutes to an event being timed.

```
66696 #include <time.h>
66697 #include <stdio.h>
66698 ...
66699 time_t now;
66700 int minutes_to_event;
66701 ...
66702 time(&now);
66703 minutes_to_event = ...;
66704 printf("The time is ");
66705 puts(asctime(localtime(&now)));
66706 printf("There are %d minutes to the event.\n",
66707     minutes_to_event);
66708 ...
```

66709 **APPLICATION USAGE**

66710 None.

66711 **RATIONALE**

66712 The *time()* function returns a value in seconds (type **time\_t**) while *times()* returns a set of values  
66713 in clock ticks (type **clock\_t**). Some historical implementations, such as 4.3 BSD, have  
66714 mechanisms capable of returning more precise times (see below). A generalized timing scheme  
66715 to unify these various timing mechanisms has been proposed but not adopted.

66716 Implementations in which **time\_t** is a 32-bit signed integer (many historical implementations)  
66717 fail in the year 2038. POSIX.1-2008 does not address this problem. However, the use of the **time\_t**  
66718 type is mandated in order to ease the eventual fix.

66719 The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C  
66720 standard.

66721 Many historical implementations (including Version 7) and the 1984 /usr/group standard use  
66722 **long** instead of **time\_t**. This volume of POSIX.1-2008 uses the latter type in order to agree with  
66723 the ISO C standard.

66724 4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

66725 **FUTURE DIRECTIONS**

66726 In a future version of this volume of POSIX.1-2008, **time\_t** is likely to be required to be capable  
66727 of representing times far in the future. Whether this will be mandated as a 64-bit type or a  
66728 requirement that a specific date in the future be representable (for example, 10000 AD) is not yet  
66729 determined. Systems purchased after the approval of this volume of POSIX.1-2008 should be  
66730 evaluated to determine whether their lifetime will extend past 2038.

66731 **SEE ALSO**

66732 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gettimeofday()*, *gmtime()*, *localtime()*, *mkttime()*, *strftime()*,  
66733 *strptime()*, *utime()*

66734 XBD **<time.h>**

66735 **CHANGE HISTORY**

66736 First released in Issue 1. Derived from Issue 1 of the SVID.

66737 **Issue 6**

66738 Extensions beyond the ISO C standard are marked.

66739 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**timer\_create()**66740 **NAME**

66741 timer\_create — create a per-process timer

66742 **SYNOPSIS**

```

66743 CX #include <signal.h>
66744 #include <time.h>
66745 int timer_create(clockid_t clockid, struct sigevent *restrict evp,
66746 timer_t *restrict timerid);

```

66747 **DESCRIPTION**

66748 The *timer\_create()* function shall create a per-process timer using the specified clock, *clock\_id*, as  
 66749 the timing base. The *timer\_create()* function shall return, in the location referenced by *timerid*, a  
 66750 timer ID of type **timer\_t** used to identify the timer in timer requests. This timer ID shall be  
 66751 unique within the calling process until the timer is deleted. The particular clock, *clock\_id*, is  
 66752 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return  
 66753 from *timer\_create()*.

66754 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the  
 66755 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page  
 66756 484) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument  
 66757 pointed to a **sigevent** structure with the *sigev\_notify* member having the value SIGEV\_SIGNAL,  
 66758 the *sigev\_signo* having a default signal number, and the *sigev\_value* member having the value of  
 66759 the timer ID.

66760 Each implementation shall define a set of clocks that can be used as timing bases for per-process  
 66761 **MON** timers. All implementations shall support a *clock\_id* of CLOCK\_REALTIME. If the Monotonic  
 66762 Clock option is supported, implementations shall support a *clock\_id* of CLOCK\_MONOTONIC.

66763 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed  
 66764 and deleted by an *exec*.

66765 **CPT** If \_POSIX\_CPUTIME is defined, implementations shall support *clock\_id* values representing the  
 66766 CPU-time clock of the calling process.

66767 **TCT** If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support *clock\_id* values  
 66768 representing the CPU-time clock of the calling thread.

66769 **CPT|TCT** It is implementation-defined whether a *timer\_create()* function will succeed if the value defined  
 66770 by *clock\_id* corresponds to the CPU-time clock of a process or thread different from the process  
 66771 or thread invoking the function.

66772 **TSA** If *evp->sigev\_notify* is SIGEV\_THREAD and *sev->sigev\_notify\_attributes* is not NULL, if the  
 66773 attribute pointed to by *sev->sigev\_notify\_attributes* has a thread stack address specified by a call  
 66774 to *pthread\_attr\_setstack()*, the results are unspecified if the signal is generated more than once.

66775 **RETURN VALUE**

66776 If the call succeeds, *timer\_create()* shall return zero and update the location referenced by *timerid*  
 66777 to a **timer\_t**, which can be passed to the per-process timer calls. If an error occurs, the function  
 66778 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if  
 66779 an error occurs.

66780 **ERRORS**66781 The *timer\_create()* function shall fail if:

66782 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

66783	[EAGAIN]	The calling process has already created all of the timers it is allowed by this implementation.
66784		
66785	[EINVAL]	The specified clock ID is not defined.
66786	CPT TCT [ENOTSUP]	The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by <i>clock_id</i> and associated with a process or thread different from the process or thread invoking <i>timer_create()</i> .
66787		
66788		

66789 **EXAMPLES**

66790 None.

66791 **APPLICATION USAGE**

66792 If a timer is created which has *evp->sigev\_sigev\_notify* set to SIGEV\_THREAD and the attribute  
 66793 pointed to by *evp->sigev\_notify\_attributes* has a thread stack address specified by a call to  
 66794 *pthread\_attr\_setstack()*, the memory dedicated as a thread stack cannot be recovered. The reason  
 66795 for this is that the threads created in response to a timer expiration are created detached, or in an  
 66796 unspecified way if the thread attribute's *detachstate* is PTHREAD\_CREATE\_JOINABLE. In  
 66797 neither case is it valid to call *pthread\_join()*, which makes it impossible to determine the lifetime  
 66798 of the created thread which thus means the stack memory cannot be reused.

66799 **RATIONALE**66800 **Periodic Timer Overrun and Resource Allocation**

66801 The specified timer facilities may deliver realtime signals (that is, queued signals) on  
 66802 implementations that support this option. Since realtime applications cannot afford to lose  
 66803 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it  
 66804 must be possible to ensure that sufficient resources exist to deliver the signal when the event  
 66805 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a  
 66806 request and a subsequent signal generation. If the request cannot allocate the signal delivery  
 66807 resources, it can fail the call with an [EAGAIN] error.

66808 Periodic timers are a special case. A single request can generate an unspecified number of  
 66809 signals. This is not a problem if the requesting process can service the signals as fast as they are  
 66810 generated, thus making the signal delivery resources available for delivery of subsequent  
 66811 periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic  
 66812 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the  
 66813 currently pending signal has been delivered.

66814 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a  
 66815 pending signal are generated, it is implementation-defined whether a signal is delivered for each  
 66816 occurrence. This is not adequate for some realtime applications. So a mechanism is required to  
 66817 allow applications to detect how many timer expirations were delayed without requiring an  
 66818 indefinite amount of system resources to store the delayed expirations.

66819 The specified facilities provide for an overrun count. The overrun count is defined as the  
 66820 number of extra timer expirations that occurred between the time a timer expiration signal is  
 66821 generated and the time the signal is delivered. The signal-catching function, if it is concerned  
 66822 with overruns, can retrieve this count on entry. With this method, a periodic timer only needs  
 66823 one “signal queuing resource” that can be allocated at the time of the *timer\_create()* function call.

66824 A function is defined to retrieve the overrun count so that an application need not allocate static  
 66825 storage to contain the count, and an implementation need not update this storage  
 66826 asynchronously on timer expirations. But, for some high-frequency periodic applications, the  
 66827 overhead of an additional system call on each timer expiration may be prohibitive. The  
 66828 functions, as defined, permit an implementation to maintain the overrun count in user space,

**timer\_create()**

66829 associated with the *timerid*. The *timer\_getoverrun()* function can then be implemented as a macro  
 66830 that uses the *timerid* argument (which may just be a pointer to a user space structure containing  
 66831 the counter) to locate the overrun count with no system call overhead. Other implementations,  
 66832 less concerned with this class of applications, can avoid the asynchronous update of user space  
 66833 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

66834 **Timer Expiration Signal Parameters**

66835 The Realtime Signals Extension option supports an application-specific datum that is delivered  
 66836 to the extended signal handler. This value is explicitly specified by the application, along with  
 66837 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined  
 66838 value can be either an integer constant or a pointer. This explicit specification of the value, as  
 66839 opposed to always sending the timer ID, was selected based on existing practice.

66840 It is common practice for realtime applications (on non-POSIX systems or realtime extended  
 66841 POSIX systems) to use the parameters of event handlers as the case label of a switch statement or  
 66842 as a pointer to an application-defined data structure. Since *timer\_ids* are dynamically allocated  
 66843 by the *timer\_create()* function, they can be used for neither of these functions without additional  
 66844 application overhead in the signal handler; for example, to search an array of saved timer IDs to  
 66845 associate the ID with a constant or application data structure.

66846 **FUTURE DIRECTIONS**

66847 None.

66848 **SEE ALSO**

66849 [clock\\_getres\(\)](#), [timer\\_delete\(\)](#), [timer\\_getoverrun\(\)](#)

66850 XBD [<signal.h>](#), [<time.h>](#)

66851 **CHANGE HISTORY**

66852 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

66853 **Issue 6**

66854 The *timer\_create()* function is marked as part of the Timers option.

66855 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 66856 implementation does not support the Timers option.

66857 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

66858 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the  
 66859 requirement for the CLOCK\_MONOTONIC clock under the Monotonic Clock option.

66860 The **restrict** keyword is added to the *timer\_create()* prototype for alignment with the  
 66861 ISO/IEC 9899:1999 standard.

66862 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the  
 66863 DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created  
 66864 with the notification method set to SIGEV\_THREAD.

66865 **Issue 7**

66866 The *timer\_create()* function is moved from the Timers option to the Base.

66867 **NAME**

66868 timer\_delete — delete a per-process timer

66869 **SYNOPSIS**

```
66870 CX #include <time.h>
66871 int timer_delete(timer_t timerid);
```

66872 **DESCRIPTION**

66873 The *timer\_delete()* function deletes the specified timer, *timerid*, previously created by the  
 66874 *timer\_create()* function. If the timer is armed when *timer\_delete()* is called, the behavior shall be  
 66875 as if the timer is automatically disarmed before removal. The disposition of pending signals for  
 66876 the deleted timer is unspecified.

66877 **RETURN VALUE**

66878 If successful, the *timer\_delete()* function shall return a value of zero. Otherwise, the function shall  
 66879 return a value of  $-1$  and set *errno* to indicate the error.

66880 **ERRORS**

66881 The *timer\_delete()* function may fail if:

66882 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

66883 **EXAMPLES**

66884 None.

66885 **APPLICATION USAGE**

66886 None.

66887 **RATIONALE**

66888 None.

66889 **FUTURE DIRECTIONS**

66890 None.

66891 **SEE ALSO**

66892 [timer\\_create\(\)](#)

66893 XBD [<time.h>](#)

66894 **CHANGE HISTORY**

66895 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

66896 **Issue 6**

66897 The *timer\_delete()* function is marked as part of the Timers option.

66898 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 66899 implementation does not support the Timers option.

66900 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS  
 66901 section so that the [EINVAL] error becomes optional.

66902 **Issue 7**

66903 The *timer\_delete()* function is moved from the Timers option to the Base.

**timer\_getoverrun()**66904 **NAME**

66905 timer\_getoverrun, timer\_gettime, timer\_settime — per-process timers

66906 **SYNOPSIS**

```

66907 CX #include <time.h>
66908
66908 int timer_getoverrun(timer_t timerid);
66909 int timer_gettime(timer_t timerid, struct itimerspec *value);
66910 int timer_settime(timer_t timerid, int flags,
66911     const struct itimerspec *restrict value,
66912     struct itimerspec *restrict ovalue);

```

66913 **DESCRIPTION**

66914 The *timer\_gettime()* function shall store the amount of time until the specified timer, *timerid*,  
 66915 expires and the reload value of the timer into the space pointed to by the *value* argument. The  
 66916 *it\_value* member of this structure shall contain the amount of time before the timer expires, or  
 66917 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if  
 66918 the timer was armed with absolute time. The *it\_interval* member of *value* shall contain the reload  
 66919 value last set by *timer\_settime()*.

66920 The *timer\_settime()* function shall set the time until the next expiration of the timer specified by  
 66921 *timerid* from the *it\_value* member of the *value* argument and arm the timer if the *it\_value* member  
 66922 of *value* is non-zero. If the specified timer was already armed when *timer\_settime()* is called, this  
 66923 call shall reset the time until next expiration to the *value* specified. If the *it\_value* member of *value*  
 66924 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending  
 66925 expiration notifications is unspecified.

66926 If the flag *TIMER\_ABSTIME* is not set in the argument *flags*, *timer\_settime()* shall behave as if the  
 66927 time until next expiration is set to be equal to the interval specified by the *it\_value* member of  
 66928 *value*. That is, the timer shall expire in *it\_value* nanoseconds from when the call is made. If the  
 66929 flag *TIMER\_ABSTIME* is set in the argument *flags*, *timer\_settime()* shall behave as if the time  
 66930 until next expiration is set to be equal to the difference between the absolute time specified by  
 66931 the *it\_value* member of *value* and the current value of the clock associated with *timerid*. That is,  
 66932 the timer shall expire when the clock reaches the value specified by the *it\_value* member of *value*.  
 66933 If the specified time has already passed, the function shall succeed and the expiration  
 66934 notification shall be made.

66935 The reload value of the timer shall be set to the value specified by the *it\_interval* member of  
 66936 *value*. When a timer is armed with a non-zero *it\_interval*, a periodic (or repetitive) timer is  
 66937 specified.

66938 Time values that are between two consecutive non-negative integer multiples of the resolution of  
 66939 the specified timer shall be rounded up to the larger multiple of the resolution. Quantization  
 66940 error shall not cause the timer to expire earlier than the rounded time value.

66941 If the argument *ovalue* is not NULL, the *timer\_settime()* function shall store, in the location  
 66942 referenced by *ovalue*, a value representing the previous amount of time before the timer would  
 66943 have expired, or zero if the timer was disarmed, together with the previous timer reload value.  
 66944 Timers shall not expire before their scheduled time.

66945 Only a single signal shall be queued to the process for a given timer at any point in time. When a  
 66946 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun  
 66947 shall occur. When a timer expiration signal is delivered to or accepted by a process, the  
 66948 *timer\_getoverrun()* function shall return the timer expiration overrun count for the specified  
 66949 timer. The overrun count returned contains the number of extra timer expirations that occurred  
 66950 between the time the signal was generated (queued) and when it was delivered or accepted, up

66951 to but not including an implementation-defined maximum of {DELAYTIMER\_MAX}. If the  
 66952 number of such extra expirations is greater than or equal to {DELAYTIMER\_MAX}, then the  
 66953 overrun count shall be set to {DELAYTIMER\_MAX}. The value returned by *timer\_getoverrun()*  
 66954 shall apply to the most recent expiration signal delivery or acceptance for the timer. If no  
 66955 expiration signal has been delivered for the timer, the return value of *timer\_getoverrun()* is  
 66956 unspecified.

#### 66957 RETURN VALUE

66958 If the *timer\_getoverrun()* function succeeds, it shall return the timer expiration overrun count as  
 66959 explained above.

66960 If the *timer\_gettime()* or *timer\_settime()* functions succeed, a value of 0 shall be returned.

66961 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to  
 66962 indicate the error.

#### 66963 ERRORS

66964 The *timer\_settime()* function shall fail if:

66965 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than  
 66966 or equal to 1 000 million, and the *it\_value* member of that structure did not  
 66967 specify zero seconds and nanoseconds.

66968 These functions may fail if:

66969 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer\_create()*  
 66970 but not yet deleted by *timer\_delete()*.

66971 The *timer\_settime()* function may fail if:

66972 [EINVAL] The *it\_interval* member of *value* is not zero and the timer was created with  
 66973 notification by creation of a new thread (*sigev\_sigev\_notify* was  
 66974 SIGEV\_THREAD) and a fixed stack address has been set in the thread  
 66975 attribute pointed to by *sigev\_notify\_attributes*.

#### 66976 EXAMPLES

66977 None.

#### 66978 APPLICATION USAGE

66979 Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a  
 66980 new thread. Since it cannot be assumed that the thread created for one expiration is finished  
 66981 before the next expiration of the timer, it could happen that two threads use the same memory as  
 66982 a stack at the same time. This is invalid and produces undefined results.

#### 66983 RATIONALE

66984 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The  
 66985 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either  
 66986 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for  
 66987 these common rates. The granularity of practical clocks implies that if one reads a given clock  
 66988 twice in rapid succession, one may get the same time value twice; and that timers must wait for  
 66989 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too  
 66990 soon. Note also that the granularity of the clock may be significantly coarser than the resolution  
 66991 of the data format used to set and get time and interval values. Also note that some  
 66992 implementations may choose to adjust time and/or interval values to exactly match the ticks of  
 66993 the underlying clock.

66994 This volume of POSIX.1-2008 defines functions that allow an application to determine the  
 66995 implementation-supported resolution for the clocks and requires an implementation to

**timer\_getoverrun()**

66996 document the resolution supported for timers and *nanosleep()* if they differ from the supported  
66997 clock resolution. This is more of a procurement issue than a runtime application issue.

66998 **FUTURE DIRECTIONS**

66999 None.

67000 **SEE ALSO**

67001 *clock\_getres()*, *timer\_create()*

67002 XBD <**time.h**>

67003 **CHANGE HISTORY**

67004 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

67005 **Issue 6**

67006 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are marked as part of the  
67007 Timers option.

67008 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
67009 implementation does not support the Timers option.

67010 The [EINVAL] error condition is updated to include the following: “and the *it\_value* member of  
67011 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC  
67012 Interpretation 1003.1 #89.

67013 The DESCRIPTION for *timer\_getoverrun()* is updated to clarify that “If no expiration signal has  
67014 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return  
67015 value of *timer\_getoverrun()* is unspecified”.

67016 The **restrict** keyword is added to the *timer\_settime()* prototype for alignment with the  
67017 ISO/IEC 9899:1999 standard.

67018 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS  
67019 section so that the mandatory [EINVAL] error (“The *timerid* argument does not correspond to an  
67020 ID returned by *timer\_create()* but not yet deleted by *timer\_delete()*”) becomes optional.

67021 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS  
67022 section to include an optional [EINVAL] error for the case when a timer is created with the  
67023 notification method set to SIGEV\_THREAD. APPLICATION USAGE text is also added.

67024 **Issue 7**

67025 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are moved from the Timers  
67026 option to the Base.

67027 Functionality relating to the Realtime Signals Extension option is moved to the Base.

67028 **NAME**67029 `times` — get process and waited-for child process times67030 **SYNOPSIS**

```
67031 #include <sys/times.h>
67032 clock_t times(struct tms *buffer);
```

67033 **DESCRIPTION**

67034 The `times()` function shall fill the `tms` structure pointed to by `buffer` with time-accounting  
 67035 information. The `tms` structure is defined in `<sys/times.h>`.

67036 All times are measured in terms of the number of clock ticks used.

67037 The times of a terminated child process shall be included in the `tms_cutime` and `tms_cstime`  
 67038 elements of the parent when `wait()`, `waitid()`, or `waitpid()` returns the process ID of this  
 67039 terminated child. If a child process has not waited for its children, their times shall not be  
 67040 included in its times.

- 67041 • The `tms_utime` structure member is the CPU time charged for the execution of user  
 67042 instructions of the calling process.
- 67043 • The `tms_stime` structure member is the CPU time charged for execution by the system on  
 67044 behalf of the calling process.
- 67045 • The `tms_cutime` structure member is the sum of the `tms_utime` and `tms_cutime` times of the  
 67046 child processes.
- 67047 • The `tms_cstime` structure member is the sum of the `tms_stime` and `tms_cstime` times of the  
 67048 child processes.

67049 **RETURN VALUE**

67050 Upon successful completion, `times()` shall return the elapsed real time, in clock ticks, since an  
 67051 arbitrary point in the past (for example, system start-up time). This point does not change from  
 67052 one invocation of `times()` within the process to another. The return value may overflow the  
 67053 possible range of type `clock_t`. If `times()` fails, `(clock_t)-1` shall be returned and `errno` set to  
 67054 indicate the error.

67055 **ERRORS**

67056 No errors are defined.

67057 **EXAMPLES**67058 **Timing a Database Lookup**

67059 The following example defines two functions, `start_clock()` and `end_clock()`, that are used to time  
 67060 a lookup. It also defines variables of type `clock_t` and `tms` to measure the duration of  
 67061 transactions. The `start_clock()` function saves the beginning times given by the `times()` function.  
 67062 The `end_clock()` function gets the ending times and prints the difference between the two times.

```
67063 #include <sys/times.h>
67064 #include <stdio.h>
67065 ...
67066 void start_clock(void);
67067 void end_clock(char *msg);
67068 ...
67069 static clock_t st_time;
67070 static clock_t en_time;
67071 static struct tms st_cpu;
```

**times()**

```

67072     static struct tms en_cpu;
67073     ...
67074     void
67075     start_clock()
67076     {
67077         st_time = times(&st_cpu);
67078     }
67079     /* This example assumes that the result of each subtraction
67080        is within the range of values that can be represented in
67081        an integer type. */
67082     void
67083     end_clock(char *msg)
67084     {
67085         en_time = times(&en_cpu);
67086
67087         fputs(msg, stdout);
67088         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
67089             (intmax_t)(en_time - st_time),
67089             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
67090             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
67091     }

```

**APPLICATION USAGE**

67092 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per  
67093 second as it may vary from system to system.  
67094

**RATIONALE**

67095 The accuracy of the times reported is intentionally left unspecified to allow implementations  
67096 flexibility in design, from uniprocessor to multi-processor networks.  
67097

67098 The inclusion of times of child processes is recursive, so that a parent process may collect the  
67099 total times of all of its descendants. But the times of a child are only added to those of its parent  
67100 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process  
67101 can always see the total times of all its descendants; see also the discussion of the term  
67102 “realtime” in `alarm()`.

67103 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a  
67104 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual  
67105 systems that run continuously for longer than that. This volume of POSIX.1-2008 permits an  
67106 implementation to make the reference point for the returned value be the start-up time of the  
67107 process, rather than system start-up time.

67108 The term “charge” in this context has nothing to do with billing for services. The operating  
67109 system accounts for time used in this way. That information must be correct, regardless of how  
67110 that information is used.

**FUTURE DIRECTIONS**

67111 None.  
67112

**SEE ALSO**

67113 `alarm()`, `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, `waitid()`

67114 XBD [<sys/times.h>](#)

67116 **CHANGE HISTORY**

67117 First released in Issue 1. Derived from Issue 1 of the SVID.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**timezone()**67118 **NAME**

67119            timezone — difference from UTC and local standard time

67120 **SYNOPSIS**

```
67121 xSI        #include <time.h>
67122            extern long timezone;
```

67123 **DESCRIPTION**67124            Refer to [tzset\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67125 **NAME**

67126 tmpfile — create a temporary file

67127 **SYNOPSIS**

67128 #include &lt;stdio.h&gt;

67129 FILE \*tmpfile(void);

67130 **DESCRIPTION**

67131 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 67132 conflict between the requirements described here and the ISO C standard is unintentional. This  
 67133 volume of POSIX.1-2008 defers to the ISO C standard.

67134 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file  
 67135 shall be automatically deleted when all references to the file are closed. The file is opened as  
 67136 *fopen()* for update (*w+*), except that implementations may restrict the permissions, either by  
 67137 clearing the file mode bits or setting them to the value *S\_IRUSR | S\_IWUSR*.

67138 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is  
 67139 killed while it is processing a call to *tmpfile()*.

67140 An error message may be written to standard error if the stream cannot be opened.

67141 **RETURN VALUE**

67142 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is  
 67143 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

67144 **ERRORS**67145 The *tmpfile()* function shall fail if:

67146 CX [EINTR] A signal was caught during *tmpfile()*.

67147 CX [EMFILE] All file descriptors available to the process are currently open.

67148 CX [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

67149 CX [ENFILE] The maximum allowable number of files is currently open in the system.

67150 CX [ENOSPC] The directory or file system which would contain the new file cannot be  
 67151 expanded.

67152 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly  
 67153 in an object of type *off\_t*.

67154 The *tmpfile()* function may fail if:

67155 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

67156 CX [ENOMEM] Insufficient storage space is available.

67157 **EXAMPLES**67158 **Creating a Temporary File**

67159 The following example creates a temporary file for update, and returns a pointer to a stream for  
 67160 the created file in the *fp* variable.

67161 #include &lt;stdio.h&gt;

67162 ...

67163 FILE \*fp;

67164 fp = tmpfile ();

**tmpfile()**67165 **APPLICATION USAGE**

67166 It should be possible to open at least {TMP\_MAX} temporary files during the lifetime of the  
 67167 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number  
 67168 simultaneously open other than this limit and any limit on the number of open file descriptors  
 67169 or streams ({OPEN\_MAX}, {FOPEN\_MAX}, {STREAM\_MAX}).

67170 **RATIONALE**

67171 None.

67172 **FUTURE DIRECTIONS**

67173 None.

67174 **SEE ALSO**

67175 *fopen()*, *mkdtemp()*, *tmpnam()*, *unlink()*

67176 XBD <stdio.h>

67177 **CHANGE HISTORY**

67178 First released in Issue 1. Derived from Issue 1 of the SVID.

67179 **Issue 5**

67180 Large File Summit extensions are added.

67181 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
 67182 in previous issues.

67183 **Issue 6**

67184 Extensions beyond the ISO C standard are marked.

67185 The following new requirements on POSIX implementations derive from alignment with the  
 67186 Single UNIX Specification:

- 67187 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support  
 67188 large files.
- 67189 • The [EMFILE] optional error condition is added.

67190 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999  
 67191 standard.

67192 **Issue 7**

67193 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may  
 67194 restrict the permissions of the file created.

67195 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

67196 SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for  
 67197 {STREAM\_MAX} streams open.

67198 **NAME**

67199 tmpnam — create a name for a temporary file

67200 **SYNOPSIS**

```
67201 OB #include <stdio.h>
67202 char *tmpnam(char *s);
```

67203 **DESCRIPTION**

67204 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
67205 conflict between the requirements described here and the ISO C standard is unintentional. This  
67206 volume of POSIX.1-2008 defers to the ISO C standard.

67207 The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as  
67208 the name of an existing file. The function is potentially capable of generating {TMP\_MAX}  
67209 different strings, but any or all of them may already be in use by existing files and thus not be  
67210 suitable return values.

67211 The *tmpnam()* function generates a different string each time it is called from the same process,  
67212 up to {TMP\_MAX} times. If it is called more than {TMP\_MAX} times, the behavior is  
67213 implementation-defined.

67214 The implementation shall behave as if no function defined in this volume of POSIX.1-2008,  
67215 except *tmpnam()*, calls *tmpnam()*.

67216 CX The *tmpnam()* function need not be thread-safe if called with a NULL parameter.

67217 **RETURN VALUE**

67218 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can  
67219 be generated, the *tmpnam()* function shall return a null pointer.

67220 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and  
67221 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the  
67222 argument *s* is not a null pointer, it is presumed to point to an array of at least *L\_tmpnam* **chars**;  
67223 *tmpnam()* shall write its result in that array and shall return the argument as its value.

67224 **ERRORS**

67225 No errors are defined.

67226 **EXAMPLES**67227 **Generating a Filename**

67228 The following example generates a unique filename and stores it in the array pointed to by *ptr*.

```
67229 #include <stdio.h>
67230
67231 char filename[L_tmpnam+1];
67232 char *ptr;
67233
67234 ptr = tmpnam(filename);
```

67234 **APPLICATION USAGE**

67235 This function only creates filenames. It is the application's responsibility to create and remove  
67236 the files.

67237 Between the time a pathname is created and the file is opened, it is possible for some other  
67238 process to create a file with the same name. Applications may find *tmpfile()* more useful.

67239 Applications should use the *tmpfile()*, *mkstemp()*, or *mkdtemp()* functions instead of the

**tmpnam()**

67240 obsolescent *tmpnam()* function.

**67241 RATIONALE**

67242 None.

**67243 FUTURE DIRECTIONS**

67244 The *tmpnam()* function may be removed in a future version.

**67245 SEE ALSO**

67246 *fopen()*, *open()*, *mkdtemp()*, *tmpnam()*, *tmpfile()*, *unlink()*

67247 XBD [<stdio.h>](#)

**67248 CHANGE HISTORY**

67249 First released in Issue 1. Derived from Issue 1 of the SVID.

**67250 Issue 5**

67251 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

**67252 Issue 6**

67253 Extensions beyond the ISO C standard are marked.

67254 The normative text is updated to avoid use of the term “must” for application requirements.

67255 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

67256 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the  
67257 DESCRIPTION to allow implementations of the *tmpnam()* function to call *tmpnam()*.

**67258 Issue 7**

67259 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam()* function  
67260 need not be thread-safe if called with a NULL parameter.

67261 The *tmpnam()* function is marked obsolescent.

67262 **NAME**

67263 toascii — translate an integer to a 7-bit ASCII character

67264 **SYNOPSIS**

```
67265 OB XSI #include <ctype.h>
67266 int toascii(int c);
```

67267 **DESCRIPTION**67268 The *toascii()* function shall convert its argument into a 7-bit ASCII character.67269 **RETURN VALUE**67270 The *toascii()* function shall return the value (*c* &0x7f).67271 **ERRORS**

67272 No errors are returned.

67273 **EXAMPLES**

67274 None.

67275 **APPLICATION USAGE**67276 The *toascii()* function cannot be used portably in a localized application.67277 **RATIONALE**

67278 None.

67279 **FUTURE DIRECTIONS**67280 The *toascii()* function may be removed in a future version.67281 **SEE ALSO**67282 [isascii\(\)](#)67283 XBD [<ctype.h>](#)67284 **CHANGE HISTORY**

67285 First released in Issue 1. Derived from Issue 1 of the SVID.

67286 **Issue 7**67287 The *toascii()* function is marked obsolescent.

**tolower()**67288 **NAME**67289            `tolower, tolower_l` — transliterate uppercase characters to lowercase67290 **SYNOPSIS**

```
67291            #include <ctype.h>
67292            int tolower(int c);
67293 CX        int tolower_l(int c, locale_t locale);
```

67294 **DESCRIPTION**

67295 CX        For `tolower()`: The functionality described on this reference page is aligned with the ISO C  
67296 standard. Any conflict between the requirements described here and the ISO C standard is  
67297 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

67298 CX        The `tolower()` and `tolower_l()` functions have as a domain a type **int**, the value of which is  
67299 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the  
67300 CX        behavior is undefined. If the argument of `tolower()` or `tolower_l()` represents an uppercase letter,  
67301 and there exists a corresponding lowercase letter as defined by character type information in the  
67302 CX        program locale or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the  
67303 result shall be the corresponding lowercase letter. All other arguments in the domain are  
67304 returned unchanged.

67305 **RETURN VALUE**

67306 CX        Upon successful completion, the `tolower()` and `tolower_l()` functions shall return the lowercase  
67307 letter corresponding to the argument passed; otherwise, they shall return the argument  
67308 unchanged.

67309 **ERRORS**67310            The `tolower_l()` function may fail if:

67311 CX        **[EINVAL]** `locale` is not a valid locale object handle.

67312 **EXAMPLES**

67313            None.

67314 **APPLICATION USAGE**

67315            None.

67316 **RATIONALE**

67317            None.

67318 **FUTURE DIRECTIONS**

67319            None.

67320 **SEE ALSO**67321            [`setlocale\(\)`](#), [`uselocale\(\)`](#)67322            XBD Chapter 7 (on page 135), [`<ctype.h>`](#), [`<locale.h>`](#)67323 **CHANGE HISTORY**

67324            First released in Issue 1. Derived from Issue 1 of the SVID.

67325 **Issue 6**

67326            Extensions beyond the ISO C standard are marked.

67327 **Issue 7**

67328            The `tolower_l()` function is added from The Open Group Technical Standard, 2006, Extended API  
67329 Set Part 4.

67330 **NAME**

67331            toupper, toupper\_l — transliterate lowercase characters to uppercase

67332 **SYNOPSIS**

67333            #include &lt;ctype.h&gt;

67334            int toupper(int c);

67335 CX         int toupper\_l(int c, locale\_t locale);

67336 **DESCRIPTION**67337 CX         For *toupper()*: The functionality described on this reference page is aligned with the ISO C  
67338 standard. Any conflict between the requirements described here and the ISO C standard is  
67339 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.67340 CX         The *toupper()* and *toupper\_l()* functions have as a domain a type **int**, the value of which is  
67341 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the  
67342 behavior is undefined.67343 CX         If the argument of *toupper()* or *toupper\_l()* represents a lowercase letter, and there exists a  
67344 CX         corresponding uppercase letter as defined by character type information in the program locale  
67345 or in the locale represented by *locale*, respectively (category **LC\_CTYPE**), the result shall be the  
67346 corresponding uppercase letter.

67347            All other arguments in the domain are returned unchanged.

67348 **RETURN VALUE**67349 CX         Upon successful completion, *toupper()* and *toupper\_l()* shall return the uppercase letter  
67350 corresponding to the argument passed; otherwise, they shall return the argument unchanged.67351 **ERRORS**67352            The *toupper\_l()* function may fail if:67353 CX         [EINVAL] *locale* is not a valid locale object handle.67354 **EXAMPLES**

67355            None.

67356 **APPLICATION USAGE**

67357            None.

67358 **RATIONALE**

67359            None.

67360 **FUTURE DIRECTIONS**

67361            None.

67362 **SEE ALSO**67363            [setlocale\(\)](#), [uselocale\(\)](#)67364            XBD Chapter 7 (on page 135), [<ctype.h>](#), [<locale.h>](#)67365 **CHANGE HISTORY**

67366            First released in Issue 1. Derived from Issue 1 of the SVID.

67367 **Issue 6**

67368            Extensions beyond the ISO C standard are marked.

## toupper()

67369 **Issue 7**

67370 SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section.

67371 The *toupper\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
67372 Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67373 **NAME**67374 `towctrans, towctrans_l` — wide-character transliteration67375 **SYNOPSIS**

```
67376 #include <wctype.h>
67377
67377 wint_t towctrans(wint_t wc, wctrans_t desc);
67378 CX wint_t towctrans_l(wint_t wc, wctrans_t desc,
67379 locale_t locale);
```

67380 **DESCRIPTION**

67381 CX For `towctrans()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

67384 CX The `towctrans()` and `towctrans_l()` functions shall transliterate the wide-character code `wc` using the mapping described by `desc`.

67386 CX The current setting of the `LC_CTYPE` category in the current locale of the process or in the locale represented by `locale`, respectively, should be the same as during the call to `wctrans()` or `wctrans_l()` that returned the value `desc`.

67389 If the value of `desc` is invalid (that is, not obtained by a call to `wctrans()` or `desc` is invalidated by a subsequent call to `setlocale()` that has affected category `LC_CTYPE`), the result is unspecified.

67391 CX If the value of `desc` is invalid (that is, not obtained by a call to `wctrans_l()` with the same locale object `locale`) the result is unspecified.

67393 CX An application wishing to check for error situations should set `errno` to 0 before calling `towctrans()` or `towctrans_l()`.

67395 If `errno` is non-zero on return, an error has occurred.

67396 **RETURN VALUE**

67397 CX If successful, the `towctrans()` and `towctrans_l()` functions shall return the mapped value of `wc` using the mapping described by `desc`. Otherwise, they shall return `wc` unchanged.

67399 **ERRORS**

67400 These functions may fail if:

67401 CX [EINVAL] `desc` contains an invalid transliteration descriptor.

67402 The `towctrans_l()` function may fail if:

67403 CX [EINVAL] `locale` is not a valid locale object handle.

67404 **EXAMPLES**

67405 None.

67406 **APPLICATION USAGE**

67407 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the table below, the functions in the left column are equivalent to the functions in the right column.

67409	<code>towlower(wc)</code>	<code>towctrans(wc, wctrans("tolower"))</code>
67410	<code>towlower_l(wc, locale)</code>	<code>towctrans_l(wc, wctrans("tolower"), locale)</code>
67411	<code>toupper(wc)</code>	<code>towctrans(wc, wctrans("toupper"))</code>
67412	<code>toupper_l(wc, locale)</code>	<code>towctrans_l(wc, wctrans("toupper"), locale)</code>

**towctrans()**

System Interfaces

67413 **RATIONALE**

67414 None.

67415 **FUTURE DIRECTIONS**

67416 None.

67417 **SEE ALSO**67418 *tolower()*, *towupper()*, *wctrans()*67419 XBD <*wctype.h*>67420 **CHANGE HISTORY**

67421 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

67422 **Issue 6**

67423 Extensions beyond the ISO C standard are marked.

67424 **Issue 7**67425 The *towctrans\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
67426 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67427 **NAME**67428            `tolower`, `tolower_l` — transliterate uppercase wide-character code to lowercase67429 **SYNOPSIS**67430            `#include <wctype.h>`67431            `wint_t tolower(wint_t wc);`67432 CX        `wint_t tolower_l(wint_t wc, locale_t locale);`67433 **DESCRIPTION**67434 CX        For `tolower()`: The functionality described on this reference page is aligned with the ISO C  
67435 standard. Any conflict between the requirements described here and the ISO C standard is  
67436 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.67437 CX        The `tolower()` and `tolower_l()` functions have as a domain a type `wint_t`, the value of which  
67438 the application shall ensure is a character representable as a `wchar_t`, and a wide-character code  
67439 corresponding to a valid character in the current locale or the value of WEOF. If the argument  
67440 CX        has any other value, the behavior is undefined. If the argument of `tolower()` or `tolower_l()`  
67441 represents an uppercase wide-character code, and there exists a corresponding lowercase wide-  
67442 CX        character code as defined by character type information in the locale of the process or in the  
67443 locale represented by `locale`, respectively (category `LC_CTYPE`), the result shall be the  
67444 corresponding lowercase wide-character code. All other arguments in the domain are returned  
67445 unchanged.67446 **RETURN VALUE**67447 CX        Upon successful completion, the `tolower()` and `tolower_l()` functions shall return the  
67448 lowercase letter corresponding to the argument passed; otherwise, they shall return the  
67449 argument unchanged.67450 **ERRORS**67451            The `tolower_l()` function may fail if:67452 CX        `[EINVAL]` `locale` is not a valid locale object handle.67453 **EXAMPLES**

67454            None.

67455 **APPLICATION USAGE**

67456            None.

67457 **RATIONALE**

67458            None.

67459 **FUTURE DIRECTIONS**

67460            None.

67461 **SEE ALSO**67462            [setlocale\(\)](#), [uselocale\(\)](#)67463            XBD Chapter 7 (on page 135), [<locale.h>](#), [<wctype.h>](#)67464 **CHANGE HISTORY**

67465            First released in Issue 4.

**tolower()**67466 **Issue 5**

67467 The following change has been made in this version for alignment with  
67468 ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- 67469 • The SYNOPSIS has been changed to indicate that this function and associated data types  
67470 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

67471 **Issue 6**

67472 The normative text is updated to avoid use of the term “must” for application requirements.

67473 **Issue 7**

67474 The `tolower_l()` function is added from The Open Group Technical Standard, 2006, Extended  
67475 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67476 **NAME**

67477 towupper, towupper\_l — transliterate lowercase wide-character code to uppercase

67478 **SYNOPSIS**

67479 #include &lt;wctype.h&gt;

67480 wint\_t towupper(wint\_t wc);

67481 CX wint\_t towupper\_l(wint\_t wc, locale\_t locale);

67482 **DESCRIPTION**67483 CX For *towupper()*: The functionality described on this reference page is aligned with the ISO C  
67484 standard. Any conflict between the requirements described here and the ISO C standard is  
67485 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.67486 CX The *towupper()* and *towupper\_l()* functions have as a domain a type **wint\_t**, the value of which  
67487 the application shall ensure is a character representable as a **wchar\_t**, and a wide-character code  
67488 corresponding to a valid character in the current locale or the value of WEOF. If the argument  
67489 CX has any other value, the behavior is undefined. If the argument of *towupper()* or *towupper\_l()*  
67490 represents a lowercase wide-character code, and there exists a corresponding uppercase wide-  
67491 CX character code as defined by character type information in the locale of the process or in the  
67492 locale represented by *locale*, respectively (category *LC\_CTYPE*), the result shall be the  
67493 corresponding uppercase wide-character code. All other arguments in the domain are returned  
67494 unchanged.67495 **RETURN VALUE**67496 CX Upon successful completion, the *towupper()* and *towupper\_l()* functions shall return the  
67497 uppercase letter corresponding to the argument passed. Otherwise, they shall return the  
67498 argument unchanged.67499 **ERRORS**67500 The *towupper\_l()* function may fail if:67501 CX [EINVAL] *locale* is not a valid locale object handle.67502 **EXAMPLES**

67503 None.

67504 **APPLICATION USAGE**

67505 None.

67506 **RATIONALE**

67507 None.

67508 **FUTURE DIRECTIONS**

67509 None.

67510 **SEE ALSO**67511 [setlocale\(\)](#), [uselocale\(\)](#)67512 XBD Chapter 7 (on page 135), [<locale.h>](#), [<wctype.h>](#)67513 **CHANGE HISTORY**

67514 First released in Issue 4.

**towupper()**

System Interfaces

67515 **Issue 5**

67516 The following change has been made in this version for alignment with  
67517 ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- 67518 • The SYNOPSIS has been changed to indicate that this function and associated data types  
67519 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

67520 **Issue 6**

67521 The normative text is updated to avoid use of the term “must” for application requirements.

67522 **Issue 7**

67523 The `towupper_l()` function is added from The Open Group Technical Standard, 2006, Extended  
67524 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67525 **NAME**67526 `trunc, truncf, trunc1` — round to truncated integer value67527 **SYNOPSIS**

```
67528     #include <math.h>
67529     double trunc(double x);
67530     float truncf(float x);
67531     long double trunc1(long double x);
```

67532 **DESCRIPTION**

67533 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 67534 conflict between the requirements described here and the ISO C standard is unintentional. This  
 67535 volume of POSIX.1-2008 defers to the ISO C standard.

67536 These functions shall round their argument to the integer value, in floating format, nearest to but  
 67537 no larger in magnitude than the argument.

67538 **RETURN VALUE**

67539 Upon successful completion, these functions shall return the truncated integer value.

67540 **MX** If  $x$  is NaN, a NaN shall be returned.

67541 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

67542 **ERRORS**

67543 No errors are defined.

67544 **EXAMPLES**

67545 None.

67546 **APPLICATION USAGE**

67547 None.

67548 **RATIONALE**

67549 None.

67550 **FUTURE DIRECTIONS**

67551 None.

67552 **SEE ALSO**

67553 XBD [<math.h>](#)

67554 **CHANGE HISTORY**

67555 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**truncate()**67556 **NAME**

67557 truncate — truncate a file to a specified length

67558 **SYNOPSIS**

67559 #include &lt;unistd.h&gt;

67560 int truncate(const char \*path, off\_t length);

67561 **DESCRIPTION**67562 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be  
67563 equal to *length* bytes.67564 If the file previously was larger than *length*, the extra data is discarded. If the file was previously  
67565 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

67566 The application shall ensure that the process has write permission for the file.

67567 XSI If the request would cause the file size to exceed the soft file size limit for the process, the  
67568 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.67569 The *truncate()* function shall not modify the file offset for any open file descriptions associated  
67570 with the file. Upon successful completion, if the file size is changed, *truncate()* shall mark for  
67571 update the last data modification and last file status change timestamps of the file, and the  
67572 S\_ISUID and S\_ISGID bits of the file mode may be cleared.67573 **RETURN VALUE**67574 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*  
67575 set to indicate the error.67576 **ERRORS**67577 The *truncate()* function shall fail if:

67578 [EINTR] A signal was caught during execution.

67579 [EINVAL] The *length* argument was less than 0.

67580 [EFBIG] or [EINVAL]

67581 The *length* argument was greater than the maximum file size.

67582 [EIO] An I/O error occurred while reading from or writing to a file system.

67583 [EACCES] A component of the path prefix denies search permission, or write permission  
67584 is denied on the file.

67585 [EISDIR] The named file is a directory.

67586 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
67587 argument.

67588 [ENAMETOOLONG]

67589 The length of a component of a pathname is longer than {NAME\_MAX}.

67590 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.67591 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument  
67592 contains at least one non-*<slash>* character and ends with one or more trailing  
67593 *<slash>* characters and the last pathname component names an existing file  
67594 that is neither a directory nor a symbolic link to a directory.

67595 [EROFS] The named file resides on a read-only file system.

- 67596 The *truncate()* function may fail if:
- 67597 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
67598 resolution of the *path* argument.
- 67599 [ENAMETOOLONG]  
67600 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
67601 symbolic link produced an intermediate result with a length that exceeds  
67602 {PATH\_MAX}.
- 67603 **EXAMPLES**  
67604 None.
- 67605 **APPLICATION USAGE**  
67606 None.
- 67607 **RATIONALE**  
67608 None.
- 67609 **FUTURE DIRECTIONS**  
67610 None.
- 67611 **SEE ALSO**  
67612 *open()*  
67613 XBD <*unistd.h*>
- 67614 **CHANGE HISTORY**  
67615 First released in Issue 4, Version 2.
- 67616 **Issue 5**  
67617 Moved from X/OPEN UNIX extension to BASE.  
67618 Large File Summit extensions are added.
- 67619 **Issue 6**  
67620 This reference page is split out from the *ftruncate()* reference page.  
67621 The normative text is updated to avoid use of the term “must” for application requirements.  
67622 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
67623 [ELOOP] error condition is added.
- 67624 **Issue 7**  
67625 Austin Group Interpretation 1003.1-2001 #143 is applied.  
67626 The *truncate()* function is moved from the XSI option to the Base.  
67627 Changes are made related to support for finegrained timestamps.  
67628 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
67629 pathname exists but is not a directory or a symbolic link to a directory.

**truncf()**

67630 **NAME**  
67631 `truncf, trunc1` — round to truncated integer value

67632 **SYNOPSIS**  
67633 `#include <math.h>`  
67634 `float truncf(float x);`  
67635 `long double trunc1(long double x);`

67636 **DESCRIPTION**  
67637 Refer to *trunc()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***tsearch()**67638 **NAME**67639 `tsearch` — search a binary search tree67640 **SYNOPSIS**

```
67641 XSI #include <search.h>
67642 void *tsearch(const void *key, void **rootp,
67643 int (*compar)(const void *, const void *));
```

67644 **DESCRIPTION**67645 Refer to *tdelete()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ttyname()**67646 **NAME**67647 `ttyname, ttyname_r` — find the pathname of a terminal67648 **SYNOPSIS**

```
67649 #include <unistd.h>
67650 char *ttyname(int fildev);
67651 int ttyname_r(int fildev, char *name, size_t namesize);
```

67652 **DESCRIPTION**

67653 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname  
 67654 of the terminal associated with file descriptor *fildev*. The return value may point to static data  
 67655 whose content is overwritten by each call.

67656 The `ttyname()` function need not be thread-safe.

67657 The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated  
 67658 with the file descriptor *fildev* in the character array referenced by *name*. The array is *namesize*  
 67659 characters long and should have space for the name and the terminating null character. The  
 67660 maximum length of the terminal name shall be {TTY\_NAME\_MAX}.

67661 **RETURN VALUE**

67662 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null  
 67663 pointer shall be returned and *errno* set to indicate the error.

67664 If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be  
 67665 returned to indicate the error.

67666 **ERRORS**

67667 The `ttyname()` function may fail if:

- 67668 [EBADF] The *fildev* argument is not a valid file descriptor.
- 67669 [ENOTTY] The file associated with the *fildev* argument is not a terminal.

67670 The `ttyname_r()` function may fail if:

- 67671 [EBADF] The *fildev* argument is not a valid file descriptor.
- 67672 [ENOTTY] The file associated with the *fildev* argument is not a terminal.
- 67673 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 67674 including the terminating null character.

67675 **EXAMPLES**

67676 None.

67677 **APPLICATION USAGE**

67678 None.

67679 **RATIONALE**

67680 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a  
 67681 reference to an undefined term.

67682 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-  
 67683 zero value if it fails. The non-thread-safe version may return the name in a static data area that  
 67684 may be overwritten by each call.

67685 **FUTURE DIRECTIONS**

67686 None.

67687 **SEE ALSO**

67688 XBD &lt;unistd.h&gt;

67689 **CHANGE HISTORY**

67690 First released in Issue 1. Derived from Issue 1 of the SVID.

67691 **Issue 5**67692 The *ttyname\_r()* function is included for alignment with the POSIX Threads Extension.67693 A note indicating that the *ttyname()* function need not be reentrant is added to the  
67694 DESCRIPTION.67695 **Issue 6**67696 The *ttyname\_r()* function is marked as part of the Thread-Safe Functions option.67697 The following new requirements on POSIX implementations derive from alignment with the  
67698 Single UNIX Specification:

- 67699
- The statement that *errno* is set on error is added.
  - The [EBADF] and [ENOTTY] optional error conditions are added.
- 67700

67701 **Issue 7**

67702 Austin Group Interpretation 1003.1-2001 #156 is applied.

67703 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

67704 The *ttyname\_r()* function is moved from the Thread-Safe Functions option to the Base.

**twalk()**67705 **NAME**

67706 twalk — traverse a binary search tree

67707 **SYNOPSIS**

```
67708 XSI #include <search.h>
67709 void twalk(const void *root,
67710           void (*action)(const void *, VISIT, int ));
```

67711 **DESCRIPTION**67712 Refer to *tdelete()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67713 **NAME**

67714 daylight, timezone, tzname, tzset — set timezone conversion information

67715 **SYNOPSIS**

```
67716 #include <time.h>
67717 XSI extern int daylight;
67718 extern long timezone;
67719 CX extern char *tzname[2];
67720 void tzset(void);
```

67721 **DESCRIPTION**

67722 The `tzset()` function shall use the value of the environment variable `TZ` to set time conversion information used by `ctime()`, `localtime()`, `mktime()`, and `strptime()`. If `TZ` is absent from the environment, implementation-defined default timezone information shall be used.

67725 The `tzset()` function shall set the external variable `tzname` as follows:

```
67726 tzname[0] = "std";
67727 tzname[1] = "dst";
```

67728 where `std` and `dst` are as described in XBD Chapter 8 (on page 173).

67729 XSI The `tzset()` function also shall set the external variable `daylight` to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable `timezone` shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

67733 **RETURN VALUE**

67734 The `tzset()` function shall not return a value.

67735 **ERRORS**

67736 No errors are defined.

67737 **EXAMPLES**

67738 Example `TZ` variables and their timezone differences are given in the table below:

<i>TZ</i>	<i>timezone</i>
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	-9*60*60
MET-1MEST	-1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

67746 **APPLICATION USAGE**

67747 None.

67748 **RATIONALE**

67749 None.

67750 **FUTURE DIRECTIONS**

67751 None.

67752 **SEE ALSO**

67753 `ctime()`, `localtime()`, `mktime()`, `strptime()`

67754 XBD Chapter 8 (on page 173), `<time.h>`

**tzset()**

67755 **CHANGE HISTORY**

67756 First released in Issue 1. Derived from Issue 1 of the SVID.

67757 **Issue 6**

67758 The example is corrected.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67759 **NAME**67760 `ulimit` — get and set process limits67761 **SYNOPSIS**

```
67762 OB XSI #include <ulimit.h>
67763 long ulimit(int cmd, ...);
```

67764 **DESCRIPTION**

67765 The `ulimit()` function shall control process limits. The process limits that can be controlled by  
 67766 this function include the maximum size of a single file that can be written (this is equivalent to  
 67767 using `setrlimit()` with `RLIMIT_FSIZE`). The `cmd` values, defined in `<ulimit.h>`, include:

67768 **UL\_GETFSIZE** Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in  
 67769 units of 512-byte blocks and shall be inherited by child processes. Files of any  
 67770 size can be read. The return value shall be the integer part of the soft file size  
 67771 limit divided by 512. If the result cannot be represented as a **long**, the result is  
 67772 unspecified.

67773 **UL\_SETFSIZE** Set the file size limit for output operations of the process to the value of the  
 67774 second argument, taken as a **long**, multiplied by 512. If the result would  
 67775 overflow an **rlim\_t**, the actual value set is unspecified. Any process may  
 67776 decrease its own limit, but only a process with appropriate privileges may  
 67777 increase the limit. The return value shall be the integer part of the new file size  
 67778 limit divided by 512.

67779 The `ulimit()` function shall not change the setting of `errno` if successful.

67780 As all return values are permissible in a successful situation, an application wishing to check for  
 67781 error situations should set `errno` to 0, then call `ulimit()`, and, if it returns `-1`, check to see if `errno` is  
 67782 non-zero.

67783 **RETURN VALUE**

67784 Upon successful completion, `ulimit()` shall return the value of the requested limit. Otherwise, `-1`  
 67785 shall be returned and `errno` set to indicate the error.

67786 **ERRORS**

67787 The `ulimit()` function shall fail and the limit shall be unchanged if:

67788 [EINVAL] The `cmd` argument is not valid.

67789 [EPERM] A process not having appropriate privileges attempts to increase its file size  
 67790 limit.

67791 **EXAMPLES**

67792 None.

67793 **APPLICATION USAGE**

67794 Since the `ulimit()` function uses type **long** rather than **rlim\_t**, this function is not sufficient for file  
 67795 sizes on many current systems. Applications should use the `getrlimit()` or `setrlimit()` functions  
 67796 instead of the obsolescent `ulimit()` function.

67797 **RATIONALE**

67798 None.

**ulimit()**67799 **FUTURE DIRECTIONS**

67800 The *ulimit()* function may be removed in a future version.

67801 **SEE ALSO**

67802 *exec*, *getrlimit()*, *write()*

67803 XBD [<ulimit.h>](#)

67804 **CHANGE HISTORY**

67805 First released in Issue 1. Derived from Issue 1 of the SVID.

67806 **Issue 5**

67807 In the description of UL\_SETFSIZE, the text is corrected to refer to **rlim\_t** rather than the  
67808 spurious **rlimit\_t**.

67809 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

67810 **Issue 7**

67811 The *ulimit()* function is marked obsolescent.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67812 **NAME**

67813 umask — set and get the file mode creation mask

67814 **SYNOPSIS**

67815 #include &lt;sys/stat.h&gt;

67816 mode\_t umask(mode\_t *cmask*);67817 **DESCRIPTION**

67818 The *umask()* function shall set the file mode creation mask of the process to *cmask* and return the  
 67819 previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the  
 67820 meaning of the other bits is implementation-defined.

67821 The file mode creation mask of the process is used to turn off permission bits in the *mode*  
 67822 argument supplied during calls to the following functions:

- 67823 • *open()*, *openat()*, *creat()*, *mkdir()*, *mkdirat()*, *mkfifo()*, and *mkfifoat()*
- 67824 XSI • *mknod()*, *mknodat()*
- 67825 MSG • *mq\_open()*
- 67826 • *sem\_open()*

67827 Bit positions that are set in *cmask* are cleared in the mode of the created file.

67828 **RETURN VALUE**

67829 The file permission bits in the value returned by *umask()* shall be the previous value of the file  
 67830 mode creation mask. The state of any other bits in that value is unspecified, except that a  
 67831 subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the  
 67832 same as its state before the first call, including any unspecified use of those bits.

67833 **ERRORS**

67834 No errors are defined.

67835 **EXAMPLES**

67836 None.

67837 **APPLICATION USAGE**

67838 None.

67839 **RATIONALE**

67840 Unsigned argument and return types for *umask()* were proposed. The return type and the  
 67841 argument were both changed to **mode\_t**.

67842 Historical implementations have made use of additional bits in *cmask* for their implementation-  
 67843 defined purposes. The addition of the text that the meaning of other bits of the field is  
 67844 implementation-defined permits these implementations to conform to this volume of  
 67845 POSIX.1-2008.

67846 **FUTURE DIRECTIONS**

67847 None.

67848 **SEE ALSO**67849 *creat()*, *exec*, *mkdir()*, *mkfifo()*, *mknod()*, *mq\_open()*, *open()*, *sem\_open()*

67850 XBD &lt;sys/stat.h&gt;, &lt;sys/types.h&gt;

**umask()**67851 **CHANGE HISTORY**

67852 First released in Issue 1. Derived from Issue 1 of the SVID.

67853 **Issue 6**67854 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.67855 The following new requirements on POSIX implementations derive from alignment with the  
67856 Single UNIX Specification:

- 67857
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
67858 required for conforming implementations of previous POSIX specifications, it was not  
67859 required for UNIX applications.

67860 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the `mknod()`,  
67861 `mq_open()`, and `sem_open()` functions to the DESCRIPTION and SEE ALSO sections.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67862 **NAME**67863        **uname** — get the name of the current system67864 **SYNOPSIS**

67865        #include &lt;sys/utsname.h&gt;

67866        int uname(struct utsname \*name);

67867 **DESCRIPTION**67868        The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.67870        The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.67871        The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

67876        The format of each member is implementation-defined.

67877 **RETURN VALUE**67878        Upon successful completion, a non-negative value shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.67880 **ERRORS**

67881        No errors are defined.

67882 **EXAMPLES**

67883        None.

67884 **APPLICATION USAGE**67885        The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.67887 **RATIONALE**67888        The values of the structure members are not constrained to have any relation to the version of this volume of POSIX.1-2008 implemented in the operating system. An application should instead depend on `_POSIX_VERSION` and related constants defined in <unistd.h>.67891        This volume of POSIX.1-2008 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.67895        The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.67898        4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.67902 **FUTURE DIRECTIONS**

67903        None.

## **uname()**

67904 **SEE ALSO**

67905 XBD [<sys/utsname.h>](#)

67906 **CHANGE HISTORY**

67907 First released in Issue 1. Derived from Issue 1 of the SVID.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67908 **NAME**

67909 ungetc — push byte back into input stream

67910 **SYNOPSIS**

67911 #include &lt;stdio.h&gt;

67912 int ungetc(int *c*, FILE \**stream*);67913 **DESCRIPTION**

67914 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 67915 conflict between the requirements described here and the ISO C standard is unintentional. This  
 67916 volume of POSIX.1-2008 defers to the ISO C standard.

67917 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back  
 67918 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by  
 67919 subsequent reads on that stream in the reverse order of their pushing. A successful intervening  
 67920 call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or  
 67921 *rewind()*) shall discard any pushed-back bytes for the stream. The external storage  
 67922 corresponding to the stream shall be unchanged.

67923 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream  
 67924 without an intervening read or file-positioning operation on that stream, the operation may fail.

67925 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall  
 67926 be left unchanged.

67927 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the  
 67928 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be  
 67929 the same as it was before the bytes were pushed back. The file-position indicator is decremented  
 67930 by each successful call to *ungetc()*; if its value was 0 before a call, its value is unspecified after  
 67931 the call.

67932 **RETURN VALUE**

67933 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.  
 67934 Otherwise, it shall return EOF.

67935 **ERRORS**

67936 No errors are defined.

67937 **EXAMPLES**

67938 None.

67939 **APPLICATION USAGE**

67940 None.

67941 **RATIONALE**

67942 None.

67943 **FUTURE DIRECTIONS**

67944 None.

67945 **SEE ALSO**67946 *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*

67947 XBD &lt;stdio.h&gt;

67948 **CHANGE HISTORY**

67949 First released in Issue 1. Derived from Issue 1 of the SVID.

**ungetwc()**67950 **NAME**67951 `ungetwc` — push wide-character code back into the input stream67952 **SYNOPSIS**67953 `#include <stdio.h>`67954 `#include <wchar.h>`67955 `wint_t ungetwc(wint_t wc, FILE *stream);`67956 **DESCRIPTION**67957 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
67958 conflict between the requirements described here and the ISO C standard is unintentional. This  
67959 volume of POSIX.1-2008 defers to the ISO C standard.67960 The `ungetwc()` function shall push the character corresponding to the wide-character code  
67961 specified by `wc` back onto the input stream pointed to by `stream`. The pushed-back characters  
67962 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A  
67963 successful intervening call (with the stream pointed to by `stream`) to a file-positioning function  
67964 (`fseek()`, `fsetpos()`, or `rewind()`) discards any pushed-back characters for the stream. The external  
67965 storage corresponding to the stream is unchanged.67966 At least one character of push-back shall be provided. If `ungetwc()` is called too many times on  
67967 the same stream without an intervening read or file-positioning operation on that stream, the  
67968 operation may fail.67969 If the value of `wc` equals that of the macro `WEOF`, the operation shall fail and the input stream  
67970 shall be left unchanged.67971 A successful call to `ungetwc()` shall clear the end-of-file indicator for the stream. The value of the  
67972 file-position indicator for the stream after reading or discarding all pushed-back characters shall  
67973 be the same as it was before the characters were pushed back. The file-position indicator is  
67974 decremented (by one or more) by each successful call to `ungetwc()`; if its value was 0 before a  
67975 call, its value is unspecified after the call.67976 **RETURN VALUE**67977 Upon successful completion, `ungetwc()` shall return the wide-character code corresponding to  
67978 the pushed-back character. Otherwise, it shall return `WEOF`.67979 **ERRORS**67980 The `ungetwc()` function may fail if:67981 **CX** **[EILSEQ]** An invalid character sequence is detected, or a wide-character code does not  
67982 correspond to a valid character.67983 **EXAMPLES**

67984 None.

67985 **APPLICATION USAGE**

67986 None.

67987 **RATIONALE**

67988 None.

67989 **FUTURE DIRECTIONS**

67990 None.

67991 **SEE ALSO**67992 [fseek\(\)](#), [fsetpos\(\)](#), [read\(\)](#), [rewind\(\)](#), [setbuf\(\)](#)67993 XBD [<stdio.h>](#), [<wchar.h>](#)67994 **CHANGE HISTORY**

67995 First released in Issue 4. Derived from the MSE working draft.

67996 **Issue 5**67997 The Optional Header (OH) marking is removed from [<stdio.h>](#).67998 **Issue 6**

67999 The [EILSEQ] optional error condition is marked CX.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**unlink()**

System Interfaces

68000 **NAME**68001 `unlink, unlinkat` — remove a directory entry relative to directory file descriptor68002 **SYNOPSIS**68003 `#include <unistd.h>`68004 `int unlink(const char *path);`68005 `int unlinkat(int fd, const char *path, int flag);`68006 **DESCRIPTION**

68007 The `unlink()` function shall remove a link to a file. If `path` names a symbolic link, `unlink()` shall  
 68008 remove the symbolic link named by `path` and shall not affect any file or directory named by the  
 68009 contents of the symbolic link. Otherwise, `unlink()` shall remove the link named by the pathname  
 68010 pointed to by `path` and shall decrement the link count of the file referenced by the link.

68011 When the file's link count becomes 0 and no process has the file open, the space occupied by the  
 68012 file shall be freed and the file shall no longer be accessible. If one or more processes have the file  
 68013 open when the last link is removed, the link shall be removed before `unlink()` returns, but the  
 68014 removal of the file contents shall be postponed until all references to the file are closed.

68015 The `path` argument shall not name a directory unless the process has appropriate privileges and  
 68016 the implementation supports using `unlink()` on directories.

68017 Upon successful completion, `unlink()` shall mark for update the last data modification and last  
 68018 file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last  
 68019 file status change timestamp of the file shall be marked for update.

68020 The `unlinkat()` function shall be equivalent to the `unlink()` or `rmdir()` function except in the case  
 68021 where `path` specifies a relative path. In this case the directory entry to be removed is determined  
 68022 relative to the directory associated with the file descriptor `fd` instead of the current working  
 68023 directory. If the file descriptor was opened without `O_SEARCH`, the function shall check  
 68024 whether directory searches are permitted using the current permissions of the directory  
 68025 underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function  
 68026 shall not perform the check.

68027 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 68028 in `<fcntl.h>`:

68029 `AT_REMOVEDIR`68030 Remove the directory entry specified by `fd` and `path` as a directory, not a normal file.

68031 If `unlinkat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working  
 68032 directory is used and the behavior shall be identical to a call to `unlink()` or `rmdir()` respectively,  
 68033 depending on whether or not the `AT_REMOVEDIR` bit is set in `flag`.

68034 **RETURN VALUE**

68035 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 68036 return -1 and set `errno` to indicate the error. If -1 is returned, the named file shall not be changed.

68037 **ERRORS**

68038 These functions shall fail and shall not unlink the file if:

68039 [EACCES] Search permission is denied for a component of the path prefix, or write  
 68040 permission is denied on the directory containing the directory entry to be  
 68041 removed.

68042 [EBUSY] The file named by the `path` argument cannot be unlinked because it is being  
 68043 used by the system or another process and the implementation considers this  
 68044 an error.

68045	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
68046		
68047	[ENAMETOOLONG]	
68048		The length of a component of a pathname is longer than {NAME_MAX}.
68049	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
68050	[ENOTDIR]	A component of the path prefix is not a directory, or the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
68051		
68052		
68053		
68054	[EPERM]	The file named by <i>path</i> is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using <i>unlink()</i> on directories.
68055		
68056		
68057	XSI [EPERM] or [EACCES]	
68058		The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by the <i>path</i> argument and the process does not satisfy the criteria specified in XBD Section 4.2 (on page 107).
68059		
68060		
68061	[EROFS]	The directory entry to be unlinked is part of a read-only file system.
68062		The <i>unlinkat()</i> function shall fail if:
68063	[EACCES]	<i>fd</i> was not opened with <code>O_SEARCH</code> and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
68064		
68065	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
68066		
68067	[EEXIST] or [ENOTEMPTY]	
68068		The <i>flag</i> parameter has the <code>AT_REMOVEDIR</code> bit set and the <i>path</i> argument names a directory that is not an empty directory, or there are hard links to the directory other than <code>dot</code> or a single entry in <code>dot-dot</code> .
68069		
68070		
68071	[ENOTDIR]	The <i>flag</i> parameter has the <code>AT_REMOVEDIR</code> bit set and <i>path</i> does not name a directory.
68072		
68073		These functions may fail and not unlink the file if:
68074	XSI [EBUSY]	The file named by <i>path</i> is a named STREAM.
68075	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
68076		
68077	[ENAMETOOLONG]	
68078		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
68079		
68080		
68081	[ETXTBSY]	The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.
68082		

**unlink()**

- 68083 The *unlinkat()* function may fail if:
- 68084 [EINVAL] The value of the *flag* argument is not valid.
- 68085 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
68086 file descriptor associated with a directory.

68087 **EXAMPLES**68088 **Removing a Link to a File**

68089 The following example shows how to remove a link to a file named `/home/cnd/mod1` by  
68090 removing the entry named `/modules/pass1`.

```
68091 #include <unistd.h>
68092
68092 char *path = "/modules/pass1";
68093 int status;
68094 ...
68095 status = unlink(path);
```

68096 **Checking for an Error**

68097 The following example fragment creates a temporary password lock file named **LOCKFILE**,  
68098 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for  
68099 writing, *unlink()* is used to remove the link between the file descriptor and **LOCKFILE**.

```
68100 #include <sys/types.h>
68101 #include <stdio.h>
68102 #include <fcntl.h>
68103 #include <errno.h>
68104 #include <unistd.h>
68105 #include <sys/stat.h>
68106
68106 #define LOCKFILE "/etc/ptmp"
68107
68107 int pfd; /* Integer for file descriptor returned by open call. */
68108 FILE *fpfd; /* File pointer for use in putpwent(). */
68109 ...
68110 /* Open password Lock file. If it exists, this is an error. */
68111 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
68112 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
68113     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
68114     exit(1);
68115 }
68116
68116 /* Lock file created; proceed with fdopen of lock file so that
68117 putpwent() can be used.
68118 */
68119 if ((fpfd = fdopen(pfd, "w")) == NULL) {
68120     close(pfd);
68121     unlink(LOCKFILE);
68122     exit(1);
68123 }
```

68124 **Replacing Files**

68125 The following example fragment uses *unlink()* to discard links to files, so that they can be  
 68126 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error  
 68127 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can  
 68128 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

68129 #include <sys/types.h>
68130 #include <stdio.h>
68131 #include <fcntl.h>
68132 #include <errno.h>
68133 #include <unistd.h>
68134 #include <sys/stat.h>

68135 #define LOCKFILE "/etc/ptmp"
68136 #define PASSWDFILE "/etc/passwd"
68137 #define SAVEFILE "/etc/opasswd"
68138 ...
68139 /* If no change was made, assume error and leave passwd unchanged. */
68140 if (!valid_change) {
68141     fprintf(stderr, "Could not change password for user %s\n", user);
68142     unlink(LOCKFILE);
68143     exit(1);
68144 }

68145 /* Change permissions on new password file. */
68146 chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

68147 /* Remove saved password file. */
68148 unlink(SAVEFILE);

68149 /* Save current password file. */
68150 link(PASSWDFILE, SAVEFILE);

68151 /* Remove current password file. */
68152 unlink(PASSWDFILE);

68153 /* Save new password file as current password file. */
68154 link(LOCKFILE, PASSWDFILE);

68155 /* Remove lock file. */
68156 unlink(LOCKFILE);

68157 exit(0);

```

68158 **APPLICATION USAGE**

68159 Applications should use *rmdir()* to remove a directory.

68160 **RATIONALE**

68161 Unlinking a directory is restricted to the superuser in many historical implementations for  
 68162 reasons given in *link()* (see also *rename()*).

68163 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume  
 68164 of POSIX.1-2008 does not cover the system administration concepts of mounting and  
 68165 unmounting, the description of the error was changed to “resource busy”. (This meaning is used  
 68166 by some device drivers when a second process tries to open an exclusive use device.) The  
 68167 wording is also intended to allow implementations to refuse to remove a directory if it is the root  
 68168 or current working directory of any process.

**unlink()**

68169 The standard developers reviewed TR 24715-2006 and noted that LSB-conforming  
 68170 implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change  
 68171 to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was  
 68172 considered, but decided against since it would break existing strictly conforming and  
 68173 conforming applications. Applications written for portability to both POSIX.1-2008 and the LSB  
 68174 should be prepared to handle either error code.

68175 The purpose of the *unlinkat()* function is to remove directory entries in directories other than the  
 68176 current working directory without exposure to race conditions. Any part of the path of a file  
 68177 could be changed in parallel to a call to *unlink()*, resulting in unspecified behavior. By opening a  
 68178 file descriptor for the target directory and using the *unlinkat()* function it can be guaranteed that  
 68179 the removed directory entry is located relative to the desired directory.

**FUTURE DIRECTIONS**

68180 None.  
 68181

**SEE ALSO**

68182 *close()*, *link()*, *remove()*, *rename()*, *rmdir()*, *symlink()*

68183 XBD Section 4.2 (on page 107), `<fcntl.h>`, `<unistd.h>`

**CHANGE HISTORY**

68185 First released in Issue 1. Derived from Issue 1 of the SVID.  
 68186

**Issue 5**

68187 The [EBUSY] error is added to the optional part of the ERRORS section.  
 68188

**Issue 6**

68189 The following new requirements on POSIX implementations derive from alignment with the  
 68190 Single UNIX Specification:  
 68191

- 68192 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 68193 • The [ELOOP] mandatory error condition is added.
- 68194 • A second [ENAMETOOLONG] is added as an optional error condition.
- 68195 • The [ETXTBSY] optional error condition is added.

68196 The following changes were made to align with the IEEE P1003.1a draft standard:

- 68197 • The [ELOOP] optional error condition is added.

68198 The normative text is updated to avoid use of the term “must” for application requirements.

**Issue 7**

68200 Austin Group Interpretation 1003.1-2001 #143 is applied.

68201 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for  
 68202 operations when the S\_ISVTX bit is set.

68203 Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM]  
 68204 and [EISDIR].

68205 The *unlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 68206 Set Part 2.

68207 Changes are made related to support for finegrained timestamps.

68208 Changes are made to allow a directory to be opened for searching.

68209  
68210

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**unlockpt()**68211 **NAME**68212            **unlockpt** — unlock a pseudo-terminal master/slave pair68213 **SYNOPSIS**

```
68214 XSI      #include <stdlib.h>
68215          int unlockpt(int fildev);
```

68216 **DESCRIPTION**68217            The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the master to which *fildev* refers.68219            Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a pseudo-terminal device.68221 **RETURN VALUE**68222            Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.68224 **ERRORS**68225            The *unlockpt()* function may fail if:68226            [EBADF]            The *fildev* argument is not a file descriptor open for writing.68227            [EINVAL]          The *fildev* argument is not associated with a master pseudo-terminal device.68228 **EXAMPLES**

68229            None.

68230 **APPLICATION USAGE**

68231            None.

68232 **RATIONALE**

68233            None.

68234 **FUTURE DIRECTIONS**

68235            None.

68236 **SEE ALSO**68237            *grantpt()*, *open()*, *ptsname()*

68238            XBD &lt;stdlib.h&gt;

68239 **CHANGE HISTORY**

68240            First released in Issue 4, Version 2.

68241 **Issue 5**

68242            Moved from X/OPEN UNIX extension to BASE.

68243 **Issue 6**

68244            The normative text is updated to avoid use of the term “must” for application requirements.

68245 **NAME**

68246           unsetenv — remove an environment variable

68247 **SYNOPSIS**

```
68248 CX       #include <stdlib.h>
68249       int unsetenv(const char *name);
```

68250 **DESCRIPTION**

68251       The *unsetenv()* function shall remove an environment variable from the environment of the  
68252       calling process. The *name* argument points to a string, which is the name of the variable to be  
68253       removed. The named argument shall not contain an '=' character. If the named variable does  
68254       not exist in the current environment, the environment shall be unchanged and the function is  
68255       considered to have completed successfully.

68256       If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is  
68257       undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

68258       The *unsetenv()* function need not be thread-safe.

68259 **RETURN VALUE**

68260       Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
68261       indicate the error, and the environment shall be unchanged.

68262 **ERRORS**

68263       The *unsetenv()* function shall fail if:

68264       [EINVAL]       The *name* argument is a null pointer, points to an empty string, or points to a  
68265       string containing an '=' character.

68266 **EXAMPLES**

68267       None.

68268 **APPLICATION USAGE**

68269       None.

68270 **RATIONALE**

68271       Refer to the RATIONALE section in *setenv()*.

68272 **FUTURE DIRECTIONS**

68273       None.

68274 **SEE ALSO**

68275       *getenv()*, *setenv()*

68276       XBD *<stdlib.h>*, *<sys/types.h>*

68277 **CHANGE HISTORY**

68278       First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

68279 **Issue 7**

68280       Austin Group Interpretation 1003.1-2001 #156 is applied.

**uselocale()**68281 **NAME**

68282           uselocale — use locale in current thread

68283 **SYNOPSIS**

```
68284 CX       #include <locale.h>
68285       locale_t uselocale(locale_t newloc);
```

68286 **DESCRIPTION**

68287       The *uselocale()* function shall set the current locale for the current thread to the locale  
68288       represented by *newloc*.

68289       The value for the *newloc* argument shall be one of the following:

- 68290           1. A value returned by the *newlocale()* or *duplocale()* functions
- 68291           2. The special locale object descriptor *LC\_GLOBAL\_LOCALE*
- 68292           3. **(locale\_t)0**

68293       Once the *uselocale()* function has been called to install a thread-local locale, the behavior of every  
68294       interface using data from the current locale shall be affected for the calling thread. The current  
68295       locale for other threads shall remain unchanged.

68296       If the *newloc* argument is a null pointer, the object returned is the current locale or  
68297       *LC\_GLOBAL\_LOCALE* if there has been no previous call to *uselocale()* for the current thread.

68298       If the *newloc* argument is *LC\_GLOBAL\_LOCALE*, the thread shall use the global locale  
68299       determined by the *setlocale()* function.

68300 **RETURN VALUE**

68301       The *uselocale()* function returns the locale handle from the previous call for the current thread. If  
68302       there was no such previous call, the function shall return the value *LC\_GLOBAL\_LOCALE*.

68303 **ERRORS**

68304       The *uselocale()* function may fail if:

68305       [EINVAL]       *locale* is not a valid locale object.

68306 **EXAMPLES**

68307       None.

68308 **APPLICATION USAGE**

68309       Unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale  
68310       categories only. Applications that need to install a locale which differs only in a few categories  
68311       must use *newlocale()* to change a locale object equivalent to the currently used locale and install  
68312       it.

68313 **RATIONALE**

68314       None.

68315 **FUTURE DIRECTIONS**

68316       None.

68317 **SEE ALSO**

68318       *duplocale()*, *freelocale()*, *newlocale()*, *setlocale()*

68319       XBD **<locale.h>**

68320 **CHANGE HISTORY**  
68321 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**utime()**68322 **NAME**68323 `utime` — set file access and modification times68324 **SYNOPSIS**

```
68325 OB #include <utime.h>
68326 int utime(const char *path, const struct utimbuf *times);
```

68327 **DESCRIPTION**

68328 The `utime()` function shall set the access and modification times of the file named by the `path`  
 68329 argument.

68330 If `times` is a null pointer, the access and modification times of the file shall be set to the current  
 68331 time. The effective user ID of the process shall match the owner of the file, or the process has  
 68332 write permission to the file or has appropriate privileges, to use `utime()` in this manner.

68333 If `times` is not a null pointer, `times` shall be interpreted as a pointer to a `utimbuf` structure and the  
 68334 access and modification times shall be set to the values contained in the designated structure.  
 68335 Only a process with the effective user ID equal to the user ID of the file or a process with  
 68336 appropriate privileges may use `utime()` this way.

68337 The `utimbuf` structure is defined in the `<utime.h>` header. The times in the structure `utimbuf`  
 68338 are measured in seconds since the Epoch.

68339 Upon successful completion, the `utime()` function shall mark the last file status change  
 68340 timestamp for update; see `<sys/stat.h>`.

68341 **RETURN VALUE**

68342 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and `errno` shall  
 68343 be set to indicate the error, and the file times shall not be affected.

68344 **ERRORS**

68345 The `utime()` function shall fail if:

68346 [EACCES] Search permission is denied by a component of the path prefix; or the `times`  
 68347 argument is a null pointer and the effective user ID of the process does not  
 68348 match the owner of the file, the process does not have write permission for the  
 68349 file, and the process does not have appropriate privileges.

68350 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 68351 argument.

68352 [ENAMETOOLONG]

68353 The length of a component of a pathname is longer than {NAME\_MAX}.

68354 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

68355 [ENOTDIR]

68356 A component of the path prefix is not a directory, or the `path` argument  
 68357 contains at least one non-`<slash>` character and ends with one or more trailing  
 68358 `<slash>` characters and the last pathname component names an existing file  
 that is neither a directory nor a symbolic link to a directory.

68359 [EPERM]

68360 The `times` argument is not a null pointer and the effective user ID of the calling  
 68361 process does not match the owner of the file and the calling process does not  
 have appropriate privileges.

68362 [EROFS]

The file system containing the file is read-only.

68363 The *utime()* function may fail if:

68364 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
68365 resolution of the *path* argument.

68366 [ENAMETOOLONG]

68367 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
68368 symbolic link produced an intermediate result with a length that exceeds  
68369 {PATH\_MAX}.

#### 68370 EXAMPLES

68371 None.

#### 68372 APPLICATION USAGE

68373 Since the **utimbuf** structure only contains **time\_t** variables and is not accurate to fractions of a  
68374 second, applications should use the *utimensat()* function instead of the obsolescent *utime()*  
68375 function.

#### 68376 RATIONALE

68377 The *actime* structure member must be present so that an application may set it, even though an  
68378 implementation may ignore it and not change the last data access timestamp on the file. If an  
68379 application intends to leave one of the times of a file unchanged while changing the other, it  
68380 should use *stat()* or *fstat()* to retrieve the file's *st\_atim* and *st\_mtim* parameters, set *actime* and  
68381 *modtime* in the buffer, and change one of them before making the *utime()* call.

#### 68382 FUTURE DIRECTIONS

68383 The *utime()* function may be removed in a future version.

#### 68384 SEE ALSO

68385 *fstat()*, *fstatat()*, *futimens()*

68386 XBD <sys/stat.h>, <utime.h>

#### 68387 CHANGE HISTORY

68388 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 68389 Issue 6

68390 The following new requirements on POSIX implementations derive from alignment with the  
68391 Single UNIX Specification:

- 68392 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
68393 required for conforming implementations of previous POSIX specifications, it was not  
68394 required for UNIX applications.
- 68395 • The [ELOOP] mandatory error condition is added.
- 68396 • A second [ENAMETOOLONG] is added as an optional error condition.

68397 The following changes were made to align with the IEEE P1003.1a draft standard:

- 68398 • The [ELOOP] optional error condition is added.

68399 The normative text is updated to avoid use of the term “must” for application requirements.

#### 68400 Issue 7

68401 Austin Group Interpretation 1003.1-2001 #143 is applied.

68402 The *utime()* function is marked obsolescent.

68403 Changes are made related to support for finegrained timestamps.

**utimensat()**

System Interfaces

68404 **NAME**68405 `utimensat, utimes` — set file access and modification times relative to directory file descriptor68406 **SYNOPSIS**68407 `#include <sys/stat.h>`68408 `int utimensat(int fd, const char *path, const struct timespec times[2],`  
68409 `int flag);`

68410 XSI

`#include <sys/time.h>`68411 `int utimes(const char *path, const struct timeval times[2]);`68412 **DESCRIPTION**68413 Refer to *futimens()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***va\_arg()**68414 **NAME**68415 `va_arg`, `va_copy`, `va_end`, `va_start` — handle variable argument list68416 **SYNOPSIS**

```
68417     #include <stdarg.h>
68418     type va_arg(va_list ap, type);
68419     void va_copy(va_list dest, va_list src);
68420     void va_end(va_list ap);
68421     void va_start(va_list ap, argN);
```

68422 **DESCRIPTION**68423 Refer to XBD [<stdarg.h>](#)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**fprintf()**68424 **NAME**68425 `vdprintf`, `vfprintf`, `vprintf`, `vsprintf`, `vsnprintf` — format output of a stdarg argument list68426 **SYNOPSIS**68427 `#include <stdarg.h>`68428 `#include <stdio.h>`

```

68429 CX int vdprintf(int fildes, const char *restrict format, va_list ap);
68430 int fprintf(FILE *restrict stream, const char *restrict format,
68431 va_list ap);
68432 int vprintf(const char *restrict format, va_list ap);
68433 int vsprintf(char *restrict s, size_t n, const char *restrict format,
68434 va_list ap);
68435 int vsnprintf(char *restrict s, const char *restrict format, va_list ap);

```

68436 **DESCRIPTION**

68437 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 68438 conflict between the requirements described here and the ISO C standard is unintentional. This  
 68439 volume of POSIX.1-2008 defers to the ISO C standard.

68440 CX The `vdprintf()`, `vfprintf()`, `vprintf()`, `vsprintf()`, and `vsnprintf()` functions shall be equivalent to the  
 68441 CX `dprintf()`, `fprintf()`, `printf()`, `sprintf()`, and `sfprintf()` functions respectively, except that instead of  
 68442 being called with a variable number of arguments, they are called with an argument list as  
 68443 defined by `<stdarg.h>`.

68444 These functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro,  
 68445 the value of `ap` after the return is unspecified.

68446 **RETURN VALUE**68447 Refer to `fprintf()`.68448 **ERRORS**68449 Refer to `fprintf()`.68450 **EXAMPLES**

68451 None.

68452 **APPLICATION USAGE**68453 Applications using these functions should call `va_end(ap)` afterwards to clean up.68454 **RATIONALE**

68455 None.

68456 **FUTURE DIRECTIONS**

68457 None.

68458 **SEE ALSO**68459 `fprintf()`68460 XBD `<stdarg.h>`, `<stdio.h>`68461 **CHANGE HISTORY**

68462 First released in Issue 1. Derived from Issue 1 of the SVID.

68463 **Issue 5**68464 The `vsnprintf()` function is added.

68465 **Issue 6**

68466 The *vfprintf()*, *vprintf()*, *vsprintf()*, and *vsprintf()* functions are updated for alignment with the  
68467 ISO/IEC 9899:1999 standard.

68468 **Issue 7**

68469 The *vdprintf()* function is added to complement the *dprintf()* function from The Open Group  
68470 Technical Standard, 2006, Extended API Set Part 1.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**vfscanf()**68471 **NAME**68472 `vfscanf`, `vsscanf`, `vsscanf` — format input of a `stdarg` argument list68473 **SYNOPSIS**68474 `#include <stdarg.h>`68475 `#include <stdio.h>`68476 `int vfscanf(FILE *restrict stream, const char *restrict format,`  
68477 `va_list arg);`68478 `int vsscanf(const char *restrict format, va_list arg);`68479 `int vsscanf(const char *restrict s, const char *restrict format,`68480 `va_list arg);`68481 **DESCRIPTION**68482 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
68483 conflict between the requirements described here and the ISO C standard is unintentional. This  
68484 volume of POSIX.1-2008 defers to the ISO C standard.68485 The `vsscanf()`, `vfscanf()`, and `vsscanf()` functions shall be equivalent to the `scanf()`, `fscanf()`, and  
68486 `sscanf()` functions, respectively, except that instead of being called with a variable number of  
68487 arguments, they are called with an argument list as defined in the `<stdarg.h>` header. These  
68488 functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro, the  
68489 value of `ap` after the return is unspecified.68490 **RETURN VALUE**68491 Refer to `fscanf()`.68492 **ERRORS**68493 Refer to `fscanf()`.68494 **EXAMPLES**

68495 None.

68496 **APPLICATION USAGE**68497 Applications using these functions should call `va_end(ap)` afterwards to clean up.68498 **RATIONALE**

68499 None.

68500 **FUTURE DIRECTIONS**

68501 None.

68502 **SEE ALSO**68503 `fscanf()`68504 XBD `<stdarg.h>`, `<stdio.h>`68505 **CHANGE HISTORY**

68506 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

68507 **NAME**

68508 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

68509 **SYNOPSIS**

68510 #include &lt;stdarg.h&gt;

68511 #include &lt;stdio.h&gt;

68512 #include &lt;wchar.h&gt;

68513 int vfwprintf(FILE \*restrict stream, const wchar\_t \*restrict format,  
68514 va\_list arg);68515 int vswprintf(wchar\_t \*restrict ws, size\_t n,  
68516 const wchar\_t \*restrict format, va\_list arg);

68517 int vwprintf(const wchar\_t \*restrict format, va\_list arg);

68518 **DESCRIPTION**68519 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
68520 conflict between the requirements described here and the ISO C standard is unintentional. This  
68521 volume of POSIX.1-2008 defers to the ISO C standard.68522 The *vfwprintf()*, *vsprintf()*, and *vwprintf()* functions shall be equivalent to *fprintf()*, *swprintf()*,  
68523 and *wprintf()* respectively, except that instead of being called with a variable number of  
68524 arguments, they are called with an argument list as defined by <stdarg.h>.68525 These functions shall not invoke the *va\_end* macro. However, as these functions do invoke the  
68526 *va\_arg* macro, the value of *ap* after the return is unspecified.68527 **RETURN VALUE**68528 Refer to *fprintf()*.68529 **ERRORS**68530 Refer to *fprintf()*.68531 **EXAMPLES**

68532 None.

68533 **APPLICATION USAGE**68534 Applications using these functions should call *va\_end(ap)* afterwards to clean up.68535 **RATIONALE**

68536 None.

68537 **FUTURE DIRECTIONS**

68538 None.

68539 **SEE ALSO**68540 *fprintf()*

68541 XBD &lt;stdarg.h&gt;, &lt;stdio.h&gt;, &lt;wchar.h&gt;

68542 **CHANGE HISTORY**68543 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
68544 (E).68545 **Issue 6**68546 The *vfwprintf()*, *vsprintf()*, and *vwprintf()* prototypes are updated for alignment with the  
68547 ISO/IEC 9899:1999 standard. ()

**vfwscanf()**68548 **NAME**68549 `vfwscanf`, `vswscanf`, `wscanf` — wide-character formatted input of a `stdarg` argument list68550 **SYNOPSIS**68551 `#include <stdarg.h>`68552 `#include <stdio.h>`68553 `#include <wchar.h>`68554 `int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,`  
68555 `va_list arg);`68556 `int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,`  
68557 `va_list arg);`68558 `int wscanf(const wchar_t *restrict format, va_list arg);`68559 **DESCRIPTION**68560 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
68561 conflict between the requirements described here and the ISO C standard is unintentional. This  
68562 volume of POSIX.1-2008 defers to the ISO C standard.68563 The `vfwscanf()`, `vswscanf()`, and `wscanf()` functions shall be equivalent to the `fscanf()`,  
68564 `swscanf()`, and `wscanf()` functions, respectively, except that instead of being called with a variable  
68565 number of arguments, they are called with an argument list as defined in the `<stdarg.h>` header.  
68566 These functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro,  
68567 the value of `ap` after the return is unspecified.68568 **RETURN VALUE**68569 Refer to `fscanf()`.68570 **ERRORS**68571 Refer to `fscanf()`.68572 **EXAMPLES**

68573 None.

68574 **APPLICATION USAGE**68575 Applications using these functions should call `va_end(ap)` afterwards to clean up.68576 **RATIONALE**

68577 None.

68578 **FUTURE DIRECTIONS**

68579 None.

68580 **SEE ALSO**68581 `fscanf()`68582 XBD `<stdarg.h>`, `<stdio.h>`, `<wchar.h>`68583 **CHANGE HISTORY**

68584 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

68585 **NAME**

68586 vprintf — format the output of a stdarg argument list

68587 **SYNOPSIS**

68588 #include &lt;stdarg.h&gt;

68589 #include &lt;stdio.h&gt;

68590 int vprintf(const char \*restrict *format*, va\_list *ap*);68591 **DESCRIPTION**68592 Refer to *vfprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**vscanf()***System Interfaces*68593 **NAME**68594 `vscanf` — format input of a stdarg argument list68595 **SYNOPSIS**68596 `#include <stdarg.h>`68597 `#include <stdio.h>`68598 `int vscanf(const char *restrict format, va_list arg);`68599 **DESCRIPTION**68600 Refer to *vfscanf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

68601 **NAME**

68602        vsnprintf, vsprintf — format output of a stdarg argument list

68603 **SYNOPSIS**

68604        #include &lt;stdarg.h&gt;

68605        #include &lt;stdio.h&gt;

68606        int vsnprintf(char \*restrict *s*, size\_t *n*,68607            const char \*restrict *format*, va\_list *ap*);68608        int vsprintf(char \*restrict *s*, const char \*restrict *format*,68609            va\_list *ap*);68610 **DESCRIPTION**68611        Refer to *vfprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**vsscanf()**68612 **NAME**68613 `vsscanf` — format input of a `stdarg` argument list68614 **SYNOPSIS**68615 `#include <stdarg.h>`68616 `#include <stdio.h>`68617 `int vsscanf(const char *restrict s, const char *restrict format,`  
68618 `va_list arg);`68619 **DESCRIPTION**68620 Refer to [vscanf\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

68621 **NAME**68622 `vswprintf` — wide-character formatted output of a `stdarg` argument list68623 **SYNOPSIS**68624 `#include <stdarg.h>`68625 `#include <stdio.h>`68626 `#include <wchar.h>`68627 `int vswprintf(wchar_t *restrict ws, size_t n,`  
68628 `const wchar_t *restrict format, va_list arg);`68629 **DESCRIPTION**68630 Refer to *vfprintf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**vswscanf()**68631 **NAME**68632 `vswscanf` — wide-character formatted input of a `stdarg` argument list68633 **SYNOPSIS**68634 `#include <stdarg.h>`68635 `#include <stdio.h>`68636 `#include <wchar.h>`68637 `int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,`  
68638 `va_list arg);`68639 **DESCRIPTION**68640 Refer to *[vfwscanf\(\)](#)*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

68641 **NAME**68642 `vwprintf` — wide-character formatted output of a `stdarg` argument list68643 **SYNOPSIS**68644 `#include <stdarg.h>`68645 `#include <stdio.h>`68646 `#include <wchar.h>`68647 `int vwprintf(const wchar_t *restrict format, va_list arg);`68648 **DESCRIPTION**68649 Refer to *[vfwprintf\(\)](#)*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**vwscanf()***System Interfaces*68650 **NAME**68651 `vwscanf` — wide-character formatted input of a `stdarg` argument list68652 **SYNOPSIS**68653 `#include <stdarg.h>`68654 `#include <stdio.h>`68655 `#include <wchar.h>`68656 `int vwscanf(const wchar_t *restrict format, va_list arg);`68657 **DESCRIPTION**68658 Refer to [vfwscanf\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 68659 NAME

68660 wait, waitpid — wait for a child process to stop or terminate

## 68661 SYNOPSIS

68662 #include &lt;sys/wait.h&gt;

68663 pid\_t wait(int \*stat\_loc);

68664 pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

## 68665 DESCRIPTION

68666 The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the  
 68667 caller's child processes. Various options permit status information to be obtained for child  
 68668 processes that have terminated or stopped. If status information is available for two or more  
 68669 child processes, the order in which their status is reported is unspecified.

68670 The *wait()* function shall suspend execution of the calling thread until status information for one  
 68671 of the terminated child processes of the calling process is available, or until delivery of a signal  
 68672 whose action is either to execute a signal-catching function or to terminate the process. If more  
 68673 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process,  
 68674 exactly one thread shall return the process status at the time of the target process termination. If  
 68675 status information is available prior to the call to *wait()*, return shall be immediate.

68676 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid\_t)-1** and the  
 68677 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and  
 68678 *options* arguments.

68679 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()*  
 68680 function shall only return the status of a child process from this set:

- 68681 • If *pid* is equal to **(pid\_t)-1**, *status* is requested for any child process. In this respect,  
 68682 *waitpid()* is then equivalent to *wait()*.
- 68683 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is  
 68684 requested.
- 68685 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that  
 68686 of the calling process.
- 68687 • If *pid* is less than **(pid\_t)-1**, *status* is requested for any child process whose process group  
 68688 ID is equal to the absolute value of *pid*.

68689 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the  
 68690 following flags, defined in the **<sys/wait.h>** header:

68691 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process  
 68692 specified by *pid* whose status has not been reported since it continued from a  
 68693 job control stop.

68694 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if  
 68695 *status* is not immediately available for one of the child processes specified by  
 68696 *pid*.

68697 **WUNTRACED** The status of any child processes specified by *pid* that are stopped, and whose  
 68698 status has not yet been reported since they stopped, shall also be reported to  
 68699 the requesting process.

68700 XSI If the calling process has SA\_NOCLDWAIT set or has SIGCHLD set to SIG\_IGN, and the process  
 68701 has no unwaited-for children that were transformed into zombie processes, the calling thread  
 68702 shall block until all of the children of the process containing the calling thread terminate, and  
 68703 *wait()* and *waitpid()* shall fail and set *errno* to [ECHILD].

**wait()**

68704 If *wait()* or *waitpid()* return because the status of a child process is available, these functions  
68705 shall return a value equal to the process ID of the child process. In this case, if the value of the  
68706 argument *stat\_loc* is not a null pointer, information shall be stored in the location pointed to by  
68707 *stat\_loc*. The value stored at the location pointed to by *stat\_loc* shall be 0 if and only if the status  
68708 returned is from a terminated child process that terminated by one of the following means:

- 68709 1. The process returned 0 from *main()*.
- 68710 2. The process called *\_exit()* or *exit()* with a *status* argument of 0.
- 68711 3. The process was terminated because the last thread in the process terminated.

68712 Regardless of its value, this information may be interpreted using the following macros, which  
68713 are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat\_val* argument is the  
68714 integer value pointed to by *stat\_loc*.

68715 WIFEXITED(*stat\_val*)  
68716 Evaluates to a non-zero value if *status* was returned for a child process that terminated  
68717 normally.

68718 WEXITSTATUS(*stat\_val*)  
68719 If the value of WIFEXITED(*stat\_val*) is non-zero, this macro evaluates to the low-order 8 bits  
68720 of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the child  
68721 process returned from *main()*.

68722 WIFSIGNALED(*stat\_val*)  
68723 Evaluates to a non-zero value if *status* was returned for a child process that terminated due  
68724 to the receipt of a signal that was not caught (see `<signal.h>`).

68725 WTERMSIG(*stat\_val*)  
68726 If the value of WIFSIGNALED(*stat\_val*) is non-zero, this macro evaluates to the number of  
68727 the signal that caused the termination of the child process.

68728 WIFSTOPPED(*stat\_val*)  
68729 Evaluates to a non-zero value if *status* was returned for a child process that is currently  
68730 stopped.

68731 WSTOPSIG(*stat\_val*)  
68732 If the value of WIFSTOPPED(*stat\_val*) is non-zero, this macro evaluates to the number of the  
68733 signal that caused the child process to stop.

68734 XSI WIFCONTINUED(*stat\_val*)  
68735 Evaluates to a non-zero value if *status* was returned for a child process that has continued  
68736 from a job control stop.

68737 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
68738 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a WIFSTOPPED(*stat\_val*) before  
68739 subsequent calls to *wait()* or *waitpid()* indicate WIFEXITED(*stat\_val*) as the result of an error  
68740 detected before the new process image starts executing.

68741 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
68742 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a WIFSIGNALED(*stat\_val*) if a signal is  
68743 sent to the parent's process group after *posix\_spawn()* or *posix\_spawnnp()* is called.

68744 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
68745 XSI WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros  
68746 WIFEXITED(*\*stat\_loc*), WIFSIGNALED(*\*stat\_loc*), and WIFSTOPPED(*\*stat\_loc*) shall evaluate to a  
68747 non-zero value.

68748 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 68749 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(\**stat\_loc*),  
 68750 XSI WIFSIGNALED(\**stat\_loc*), WIFSTOPPED(\**stat\_loc*), and WIFCONTINUED(\**stat\_loc*) shall  
 68751 evaluate to a non-zero value.

68752 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 68753 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the  
 68754 macros WIFEXITED(\**stat\_loc*) and WIFSIGNALED(\**stat\_loc*) shall evaluate to a non-zero value.

68755 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 68756 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,  
 68757 XSI exactly one of the macros WIFEXITED(\**stat\_loc*), WIFSIGNALED(\**stat\_loc*), and  
 68758 WIFCONTINUED(\**stat\_loc*) shall evaluate to a non-zero value.

68759 If \_POSIX\_REALTIME\_SIGNALS is defined, and the implementation queues the SIGCHLD  
 68760 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any  
 68761 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.  
 68762 Any other pending SIGCHLD signals shall remain pending.

68763 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child  
 68764 process is available, any pending SIGCHLD signal shall be cleared unless the status of another  
 68765 child process is available.

68766 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD  
 68767 signal is delivered.

68768 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*  
 68769 report *status*. This shall not occur unless the calling process or one of its child processes  
 68770 explicitly makes use of a non-standard extension. In these cases the interpretation of the  
 68771 reported *status* is implementation-defined.

68772 If a parent process terminates without waiting for all of its child processes to terminate, the  
 68773 remaining child processes shall be assigned a new parent process ID corresponding to an  
 68774 implementation-defined system process.

#### 68775 RETURN VALUE

68776 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions  
 68777 shall return a value equal to the process ID of the child process for which *status* is reported. If  
 68778 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, -1 shall be  
 68779 returned and *errno* set to [EINTR]. If *waitpid()* was invoked with WNOHANG set in *options*, it  
 68780 has at least one child process specified by *pid* for which *status* is not available, and *status* is not  
 68781 available for any process specified by *pid*, 0 is returned. Otherwise, (pid\_t)-1 shall be returned,  
 68782 and *errno* set to indicate the error.

#### 68783 ERRORS

68784 The *wait()* function shall fail if:

68785 [ECHILD] The calling process has no existing unwaited-for child processes.  
 68786 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
 68787 by *stat\_loc* is undefined.

68788 The *waitpid()* function shall fail if:

68789 [ECHILD] The process specified by *pid* does not exist or is not a child of the calling  
 68790 process, or the process group specified by *pid* does not exist or does not have  
 68791 any member process that is a child of the calling process.

**wait()**

68792 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
68793 by *stat\_loc* is undefined.

68794 [EINVAL] The *options* argument is not valid.

68795 **EXAMPLES**68796 **Waiting for a Child Process and then Checking its Status**

68797 The following example demonstrates the use of *waitpid()*, *fork()*, and the macros used to  
68798 interpret the status value returned by *waitpid()* (and *wait()*). The code segment creates a child  
68799 process which does some unspecified work. Meanwhile the parent loops performing calls to  
68800 *waitpid()* to monitor the status of the child. The loop terminates when child termination is  
68801 detected.

```
68802 #include <stdio.h>
68803 #include <stdlib.h>
68804 #include <unistd.h>
68805 #include <sys/wait.h>
68806 ...
68807 pid_t child_pid, wpid;
68808 int status;
68809
68810 child_pid = fork();
68811 if (child_pid == -1) { /* fork() failed */
68812     perror("fork");
68813     exit(EXIT_FAILURE);
68814 }
68815 if (child_pid == 0) { /* This is the child */
68816     /* Child does some work and then terminates */
68817     ...
68818 } else { /* This is the parent */
68819     do {
68820         wpid = waitpid(child_pid, &status, WUNTRACED
68821 #ifndef WCONTINUED /* Not all implementations support this */
68822 | WCONTINUED
68823 #endif
68824 );
68825 if (wpid == -1) {
68826     perror("waitpid");
68827     exit(EXIT_FAILURE);
68828 }
68829 if (WIFEXITED(status)) {
68830     printf("child exited, status=%d\n", WEXITSTATUS(status));
68831 } else if (WIFSIGNALED(status)) {
68832     printf("child killed (signal %d)\n", WTERMSIG(status));
68833 } else if (WIFSTOPPED(status)) {
68834     printf("child stopped (signal %d)\n", WSTOPSIG(status));
68835 #ifndef WIFCONTINUED /* Not all implementations support this */
68836 } else if (WIFCONTINUED(status)) {
68837     printf("child continued\n");
68838 }
```

```

68837     #endif
68838         } else { /* Non-standard case -- may never happen */
68839             printf("Unexpected status (0x%x)\n", status);
68840         }
68841     } while (!WIFEXITED(status) && !WIFSIGNALED(status));
68842 }

```

### 68843 **Waiting for a Child Process in a Signal Handler for SIGCHLD**

68844 The following example demonstrates how to use *waitpid()* in a signal handler for SIGCHLD  
68845 without passing *-1* as the *pid* argument. (See the APPLICATION USAGE section below for the  
68846 reasons why passing a *pid* of *-1* is not recommended.) The method used here relies on the  
68847 standard behavior of *waitpid()* when SIGCHLD is blocked. On historical non-conforming  
68848 systems, the status of some child processes might not be reported.

```

68849 #include <stdlib.h>
68850 #include <stdio.h>
68851 #include <signal.h>
68852 #include <sys/types.h>
68853 #include <sys/wait.h>
68854 #include <unistd.h>
68855
68856 #define CHILDREN 10
68857
68858 static void
68859 handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
68860 {
68861     int status;
68862
68863     /*
68864      * Obtain status information for the child which
68865      * caused the SIGCHLD signal and write its exit code
68866      * to stdout.
68867      */
68868     if (sinfo->si_code != CLD_EXITED)
68869     {
68870         static char msg[] = "wrong si_code\n";
68871         write(2, msg, sizeof msg - 1);
68872     }
68873     else if (waitpid(sinfo->si_pid, &status, 0) == -1)
68874     {
68875         static char msg[] = "waitpid() failed\n";
68876         write(2, msg, sizeof msg - 1);
68877     }
68878     else if (!WIFEXITED(status))
68879     {
68880         static char msg[] = "WIFEXITED was false\n";
68881         write(2, msg, sizeof msg - 1);
68882     }
68883     else
68884     {
68885         int code = WEXITSTATUS(status);
68886         char buf[2];
68887         buf[0] = '0' + code;

```

**wait()**

```

68885         buf[1] = '\n';
68886         write(1, buf, 2);
68887     }
68888 }
68889
68889 int
68890 main(void)
68891 {
68892     int i;
68893     pid_t pid;
68894     struct sigaction sa;
68895
68895     sa.sa_flags = SA_SIGINFO;
68896     sa.sa_sigaction = handle_sigchld;
68897     sigemptyset(&sa.sa_mask);
68898     if (sigaction(SIGCHLD, &sa, NULL) == -1)
68899     {
68900         perror("sigaction");
68901         exit(EXIT_FAILURE);
68902     }
68903
68903     for (i = 0; i < CHILDREN; i++)
68904     {
68905         switch (pid = fork())
68906         {
68907             case -1:
68908                 perror("fork");
68909                 exit(EXIT_FAILURE);
68910             case 0:
68911                 sleep(2);
68912                 _exit(i);
68913         }
68914     }
68915
68915     /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
68916     alarm(3);
68917     for (;;)
68918         pause();
68919 }

```

**APPLICATION USAGE**

Calls to *wait()* will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system()*). For this and other reasons it is recommended that portable applications not use *wait()*, but instead use *waitpid()*. For these same reasons, the use of *waitpid()* with a *pid* argument of *-1*, and the use of *waitid()* with the *idtype* argument set to *P\_ALL*, are also not recommended for portable applications.

**RATIONALE**

A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()* from the original parent process. Nothing in this volume of POSIX.1-2008 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild

68934 or any other process, but a process that does not use such extensions must be guaranteed to see  
68935 *status* from only its direct children.

68936 The *waitpid()* function is provided for three reasons:

- 68937 1. To support job control
- 68938 2. To permit a non-blocking version of the *wait()* function
- 68939 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without  
68940 interfering with other terminated children for which the process has not waited

68941 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The  
68942 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The  
68943 WUNTRACED flag is used only in conjunction with job control on systems supporting job  
68944 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped  
68945 processes in that implementation: processes being traced via the *ptrace()* debugging facility and  
68946 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of  
68947 POSIX.1-2008, only the second type is relevant. The name WUNTRACED was retained because  
68948 its usage is the same, even though the name is not intuitively meaningful in this context.

68949 The third reason for the *waitpid()* function is to permit independent sections of a process to  
68950 spawn and wait for children without interfering with each other. For example, the following  
68951 problem occurs in developing a portable shell, or command interpreter:

```
68952 stream = popen("/bin/true");
68953 (void) system("sleep 100");
68954 (void) pclose(stream);
```

68955 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

68956 The status values are retrieved by macros, rather than given as specific bit encodings as they are  
68957 in most historical implementations (and thus expected by existing programs). This was  
68958 necessary to eliminate a limitation on the number of signals an implementation can support that  
68959 was inherent in the traditional encodings. This volume of POSIX.1-2008 does require that a *status*  
68960 value of zero corresponds to a process calling *\_exit(0)*, as this is the most common encoding  
68961 expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

68962 These macros syntactically operate on an arbitrary integer value. The behavior is undefined  
68963 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed to  
68964 by the *stat\_loc* argument. An early proposal attempted to make this clearer by specifying each  
68965 argument as *\*stat\_loc* rather than *stat\_val*. However, that did not follow the conventions of other  
68966 specifications in this volume of POSIX.1-2008 or traditional usage. It also could have implied  
68967 that the argument to the macro must literally be *\*stat\_loc*; in fact, that value can be stored or  
68968 passed as an argument to other functions before being interpreted by these macros.

68969 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is  
68970 the *ptrace()* function. It is called by a child process and causes that child to stop and return a  
68971 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*  
68972 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*  
68973 function). Most applications do not need to concern themselves with such extensions because  
68974 they have control over what extensions they or their children use. However, applications, such  
68975 as command interpreters, that invoke arbitrary processes may see this behavior when those  
68976 arbitrary processes misuse such extensions.

68977 Implementations that support **core** file creation or other implementation-defined actions on  
68978 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to  
68979 indicate that such actions have occurred.

**wait()**

68980 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated  
68981 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*  
68982 category with respect to SIGCHLD.

68983 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the  
68984 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the  
68985 *sigwait()* family of functions.

68986 Allowing the *wait()* family of functions to behave this way permits an implementation to be able  
68987 to deal precisely with SIGCHLD signals.

68988 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the  
68989 following guarantees regardless of the queuing depth of signals in general (the list of waitable  
68990 children can hold the SIGCHLD queue):

- 68991 1. If a SIGCHLD signal handler is established via *sigaction()* without the SA\_RESETHAND  
68992 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal  
68993 will be delivered to or accepted by the process for every child process that terminates.
- 68994 2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return  
68995 immediately with status information for a child process.
- 68996 3. When SA\_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to  
68997 receive a non-null pointer to a **siginfo\_t** structure that describes a child process for which  
68998 a wait via *waitpid()* or *waitid()* will not block or fail.
- 68999 4. The *system()* function will not cause the SIGCHLD handler of a process to be called as a  
69000 result of the *fork()/exec* executed within *system()* because *system()* will accept the  
69001 SIGCHLD signal when it performs a *waitpid()* for its child process. This is a desirable  
69002 behavior of *system()* so that it can be used in a library without causing side-effects to the  
69003 application linked with the library.

69004 An implementation that does not permit the *wait()* family of functions to accept (discard) a  
69005 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the  
69006 guarantees described above for the following reasons:

#### 69007 Guarantee #1

69008 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by  
69009 the system can make this guarantee, the counter-example is the case of a process that blocks  
69010 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the  
69011 implementation supports queued signals, then eventually the system will run out of  
69012 memory for the queue. The guarantee cannot be made because there must be some limit to  
69013 the depth of queuing.

#### 69014 Guarantees #2 and #3

69015 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD  
69016 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*  
69017 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the  
69018 process has disappeared.

#### 69019 Guarantee #4

69020 Although possible to make this guarantee, *system()* would have to set the SIGCHLD  
69021 handler to SIG\_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded  
69022 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This  
69023 would have the undesirable side-effect of discarding all SIGCHLD signals pending to the  
69024 process.

69025 **FUTURE DIRECTIONS**

69026 None.

69027 **SEE ALSO**69028 *exec*, *exit()*, *fork()*, *system()*, *waitid()*69029 XBD Section 4.11 (on page 110), [<signal.h>](#), [<sys/wait.h>](#)69030 **CHANGE HISTORY**

69031 First released in Issue 1. Derived from Issue 1 of the SVID.

69032 **Issue 5**

69033 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

69034 **Issue 6**69035 The following new requirements on POSIX implementations derive from alignment with the  
69036 Single UNIX Specification:

- 69037
- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
69038 required for conforming implementations of previous POSIX specifications, it was not  
69039 required for UNIX applications.

69040 The following changes were made to align with the IEEE P1003.1a draft standard:

- 69041
- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

69042 The semantics of WIFSTOPPED(*stat\_val*), WIFEXITED(*stat\_val*), and WIFSIGNALED(*stat\_val*)  
69043 are defined with respect to *posix\_spawn()* or *posix\_spawnp()* for alignment with IEEE Std  
69044 1003.1d-1999.

69045 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

69046 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the  
69047 EXAMPLES section.69048 **Issue 7**

69049 SD5-XSH-ERN-202 is applied.

69050 APPLICATION USAGE is added, recommending that the *wait()* function not be used.69051 An additional example for *waitpid()* is added.

**waitid()**69052 **NAME**69053 `waitid` — wait for a child process to change state69054 **SYNOPSIS**69055 `#include <sys/wait.h>`69056 `int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);`69057 **DESCRIPTION**

69058 The `waitid()` function shall suspend the calling thread until one child of the process containing  
 69059 the calling thread changes state. It records the current state of a child in the structure pointed to  
 69060 by `infop`. The fields of the structure pointed to by `infop` are filled in as described for the  
 69061 SIGCHLD signal in `<signal.h>`. If a child process changed state prior to the call to `waitid()`,  
 69062 `waitid()` shall return immediately. If more than one thread is suspended in `wait()`, `waitid()`, or  
 69063 `waitpid()` waiting for termination of the same process, exactly one thread shall return the process  
 69064 status at the time of the target process termination.

69065 The `idtype` and `id` arguments are used to specify which children `waitid()` waits for.

69066 If `idtype` is P\_PID, `waitid()` shall wait for the child with a process ID equal to `(pid_t)id`.

69067 If `idtype` is P\_PGID, `waitid()` shall wait for any child with a process group ID equal to `(pid_t)id`.

69068 If `idtype` is P\_ALL, `waitid()` shall wait for any children and `id` is ignored.

69069 The `options` argument is used to specify which state changes `waitid()` shall wait for. It is formed  
 69070 by OR'ing together the following flags:

69071 WCONTINUED Status shall be returned for any child that was stopped and has been  
 69072 continued.

69073 WEXITED Wait for processes that have exited.

69074 WNOHANG Do not hang if no status is available; return immediately.

69075 WNOWAIT Keep the process whose status is returned in `infop` in a waitable state. This  
 69076 shall not affect the state of the process; the process may be waited for again  
 69077 after this call completes.

69078 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

69079 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to  
 69080 be OR'ed in with the `options` argument.

69081 The application shall ensure that the `infop` argument points to a `siginfo_t` structure. If `waitid()`  
 69082 returns because a child process was found that satisfied the conditions indicated by the  
 69083 arguments `idtype` and `options`, then the structure pointed to by `infop` shall be filled in by the  
 69084 system with the status of the process. The `si_signo` member shall always be equal to SIGCHLD.

69085 **RETURN VALUE**

69086 If WNOHANG was specified and status is not available for any process specified by `idtype` and  
 69087 `id`, 0 shall be returned. If `waitid()` returns due to the change of state of one of its children, 0 shall  
 69088 be returned. Otherwise, -1 shall be returned and `errno` set to indicate the error.

69089 **ERRORS**

69090 The `waitid()` function shall fail if:

69091 [ECHILD] The calling process has no existing unwaited-for child processes.

69092 [EINTR] The `waitid()` function was interrupted by a signal.

69093 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid  
69094 set of processes.

#### 69095 EXAMPLES

69096 None.

#### 69097 APPLICATION USAGE

69098 Calls to *waitid()* with *idtype* equal to P\_ALL will collect information about any child process.  
69099 This may result in interactions with other interfaces that may be waiting for their own children  
69100 (such as by use of *system()*). For this reason it is recommended that portable applications not  
69101 use *waitid()* with *idtype* of P\_ALL. See also APPLICATION USAGE for *wait()*.

#### 69102 RATIONALE

69103 None.

#### 69104 FUTURE DIRECTIONS

69105 None.

#### 69106 SEE ALSO

69107 *exec*, *exit()*, *wait()*

69108 XBD <signal.h>, <sys/wait.h>

#### 69109 CHANGE HISTORY

69110 First released in Issue 4, Version 2.

#### 69111 Issue 5

69112 Moved from X/OPEN UNIX extension to BASE.

69113 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 69114 Issue 6

69115 The normative text is updated to avoid use of the term “must” for application requirements.

#### 69116 Issue 7

69117 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

69118 The *waitid()* function is moved from the XSI option to the Base.

69119 APPLICATION USAGE is added, recommending that the *waitid()* function not be used with  
69120 *idtype* equal to P\_ALL.

69121 The description of the WNOHANG flag is updated.

**waitpid()***System Interfaces*69122 **NAME**

69123       waitpid — wait for a child process to stop or terminate

69124 **SYNOPSIS**

69125       #include &lt;sys/wait.h&gt;

69126       pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

69127 **DESCRIPTION**69128       Refer to *wait()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wcpcpy()**69129 **NAME**69130 `wcpcpy` — copy a wide-character string, returning a pointer to its end69131 **SYNOPSIS**69132 CX `#include <wchar.h>`69133 `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`69134 **DESCRIPTION**69135 Refer to *wcscopy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcpncpy()**

System Interfaces

69136 **NAME**69137 `wcpncpy` — copy a fixed-size wide-character string, returning a pointer to its end69138 **SYNOPSIS**69139 CX `#include <wchar.h>`69140 `wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,`  
69141 `size_t n);`69142 **DESCRIPTION**69143 Refer to *wcsncpy()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69144 **NAME**

69145 wctomb — convert a wide-character code to a character (restartable)

69146 **SYNOPSIS**

69147 #include &lt;stdio.h&gt;

69148 size\_t wctomb(char \*restrict *s*, wchar\_t *wc*, mbstate\_t \*restrict *ps*);69149 **DESCRIPTION**69150 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
69151 conflict between the requirements described here and the ISO C standard is unintentional. This  
69152 volume of POSIX.1-2008 defers to the ISO C standard.69153 If *s* is a null pointer, the *wctomb()* function shall be equivalent to the call:69154 *wctomb(buf, L'\0', ps)*69155 where *buf* is an internal buffer.69156 If *s* is not a null pointer, the *wctomb()* function shall determine the number of bytes needed to  
69157 represent the character that corresponds to the wide character given by *wc* (including any shift  
69158 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At  
69159 most {MB\_CUR\_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,  
69160 preceded by any shift sequence needed to restore the initial shift state. The resulting state  
69161 described shall be the initial conversion state.69162 If *ps* is a null pointer, the *wctomb()* function shall use its own internal **mbstate\_t** object, which is  
69163 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
69164 pointed to by *ps* shall be used to completely describe the current conversion state of the  
69165 associated character sequence. The implementation shall behave as if no function defined in this  
69166 volume of POSIX.1-2008 calls *wctomb()*.69167 CX The *wctomb()* function need not be thread-safe if called with a NULL *ps* argument.69168 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.69169 **RETURN VALUE**69170 The *wctomb()* function shall return the number of bytes stored in the array object (including any  
69171 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this  
69172 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size\_t**)-1;  
69173 the conversion state shall be undefined.69174 **ERRORS**69175 The *wctomb()* function shall fail if:

69176 [EILSEQ] An invalid wide-character code is detected.

69177 The *wctomb()* function may fail if:69178 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

**wcrtomb()**69179 **EXAMPLES**

69180 None.

69181 **APPLICATION USAGE**

69182 None.

69183 **RATIONALE**

69184 None.

69185 **FUTURE DIRECTIONS**

69186 None.

69187 **SEE ALSO**69188 *mbsinit()*, *wcsrtombs()*69189 XBD <*wchar.h*>69190 **CHANGE HISTORY**69191 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
69192 (E).69193 **Issue 6**

69194 In the DESCRIPTION, a note on using this function in a threaded application is added.

69195 Extensions beyond the ISO C standard are marked.

69196 The normative text is updated to avoid use of the term "must" for application requirements.

69197 The *wcrtomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69198 **Issue 7**69199 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcrtomb()* function  
69200 need not be thread-safe if called with a NULL *ps* argument.

69201 Austin Group Interpretation 1003.1-2001 #170 is applied.

69202 **NAME**

69203 `wscasecmp`, `wscasecmp_l`, `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character  
 69204 string comparison

69205 **SYNOPSIS**

```
69206 CX #include <wchar.h>
69207
69207 int wscasecmp(const wchar_t *ws1, const wchar_t *ws2);
69208 int wscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
69209 locale_t locale);
69210 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
69211 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
69212 size_t n, locale_t locale);
```

69213 **DESCRIPTION**

69214 The `wscasecmp()` and `wcsncasecmp()` functions are the wide-character equivalent of the  
 69215 `strcasecmp()` and `strncasecmp()` functions, respectively.

69216 The `wscasecmp()` and `wscasecmp_l()` functions shall compare, while ignoring differences in case,  
 69217 the wide-character string pointed to by `ws1` to the wide-character string pointed to by `ws2`.

69218 The `wcsncasecmp()` and `wcsncasecmp_l()` functions shall compare, while ignoring differences in  
 69219 case, not more than `n` wide-characters from the wide-character string pointed to by `ws1` to the  
 69220 wide-character string pointed to by `ws2`.

69221 When the `LC_CTIME` category of the current locale is from the POSIX locale, these functions  
 69222 shall behave as if the strings had been converted to lowercase and then a byte comparison  
 69223 performed. Otherwise, the results are unspecified.

69224 The information for `wscasecmp_l()` and `wcsncasecmp_l()` about the case of the characters comes  
 69225 from the locale represented by `locale`.

69226 **RETURN VALUE**

69227 Upon completion, the `wscasecmp()` and `wscasecmp_l()` functions shall return an integer greater  
 69228 than, equal to, or less than 0 if the wide-character string pointed to by `ws1` is, ignoring case,  
 69229 greater than, equal to, or less than the wide-character string pointed to by `ws2`, respectively.

69230 Upon completion, the `wcsncasecmp()` and `wcsncasecmp_l()` functions shall return an integer  
 69231 greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed  
 69232 to by `ws1` is, ignoring case, greater than, equal to, or less than the possibly null wide-character  
 69233 terminated string pointed to by `ws2`, respectively.

69234 No return values are reserved to indicate an error.

69235 **ERRORS**

69236 The `wscasecmp_l()` and `wcsncasecmp_l()` functions may fail if:

69237 [EINVAL] `locale` is not a valid locale object handle.

**wscasecmp()***System Interfaces*69238 **EXAMPLES**

69239 None.

69240 **APPLICATION USAGE**

69241 None.

69242 **RATIONALE**

69243 None.

69244 **FUTURE DIRECTIONS**

69245 None.

69246 **SEE ALSO**69247 *strcasecmp()*, *wscmp()*, *wcsncmp()*69248 XBD <[wchar.h](#)>69249 **CHANGE HISTORY**

69250 First released in Issue 7.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69251 **NAME**69252 `wscat` — concatenate two wide-character strings69253 **SYNOPSIS**69254 `#include <wchar.h>`69255 `wchar_t *wscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);`69256 **DESCRIPTION**

69257 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 69258 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69259 volume of POSIX.1-2008 defers to the ISO C standard.

69260 The `wscat()` function shall append a copy of the wide-character string pointed to by `ws2`  
 69261 (including the terminating null wide-character code) to the end of the wide-character string  
 69262 pointed to by `ws1`. The initial wide-character code of `ws2` shall overwrite the null wide-character  
 69263 code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is  
 69264 undefined.

69265 **RETURN VALUE**69266 The `wscat()` function shall return `ws1`; no return value is reserved to indicate an error.69267 **ERRORS**

69268 No errors are defined.

69269 **EXAMPLES**

69270 None.

69271 **APPLICATION USAGE**

69272 None.

69273 **RATIONALE**

69274 None.

69275 **FUTURE DIRECTIONS**

69276 None.

69277 **SEE ALSO**69278 [wscnecat\(\)](#)69279 XBD [<wchar.h>](#)69280 **CHANGE HISTORY**

69281 First released in Issue 4. Derived from the MSE working draft.

69282 **Issue 6**69283 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, `s1` is  
 69284 changed to `ws1`.69285 The `wscat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

**wcschr()**69286 **NAME**69287 `wcschr` — wide-character string scanning operation69288 **SYNOPSIS**69289 `#include <wchar.h>`69290 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`69291 **DESCRIPTION**

69292 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 69293 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69294 volume of POSIX.1-2008 defers to the ISO C standard.

69295 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed  
 69296 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type  
 69297 `wchar_t` and a wide-character code corresponding to a valid character in the current locale. The  
 69298 terminating null wide-character code is considered to be part of the wide-character string.

69299 **RETURN VALUE**

69300 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if  
 69301 the wide-character code is not found.

69302 **ERRORS**

69303 No errors are defined.

69304 **EXAMPLES**

69305 None.

69306 **APPLICATION USAGE**

69307 None.

69308 **RATIONALE**

69309 None.

69310 **FUTURE DIRECTIONS**

69311 None.

69312 **SEE ALSO**69313 [wcsrchr\(\)](#)69314 XBD [<wchar.h>](#)69315 **CHANGE HISTORY**

69316 First released in Issue 4. Derived from the MSE working draft.

69317 **Issue 6**

69318 The normative text is updated to avoid use of the term “must” for application requirements.

69319 **NAME**69320 `wscmp` — compare two wide-character strings69321 **SYNOPSIS**69322 `#include <wchar.h>`69323 `int wscmp(const wchar_t *ws1, const wchar_t *ws2);`69324 **DESCRIPTION**

69325 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 69326 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69327 volume of POSIX.1-2008 defers to the ISO C standard.

69328 The `wscmp()` function shall compare the wide-character string pointed to by `ws1` to the wide-  
 69329 character string pointed to by `ws2`.

69330 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 69331 values of the first pair of wide-character codes that differ in the objects being compared.

69332 **RETURN VALUE**

69333 Upon completion, `wscmp()` shall return an integer greater than, equal to, or less than 0, if the  
 69334 wide-character string pointed to by `ws1` is greater than, equal to, or less than the wide-character  
 69335 string pointed to by `ws2`, respectively.

69336 **ERRORS**

69337 No errors are defined.

69338 **EXAMPLES**

69339 None.

69340 **APPLICATION USAGE**

69341 None.

69342 **RATIONALE**

69343 None.

69344 **FUTURE DIRECTIONS**

69345 None.

69346 **SEE ALSO**69347 `wscasecmp()`, `wcsncmp()`69348 XBD `<wchar.h>`69349 **CHANGE HISTORY**

69350 First released in Issue 4. Derived from the MSE working draft.

**wscoll()**69351 **NAME**69352 `wscoll`, `wscoll_l` — wide-character string comparison using collating information69353 **SYNOPSIS**69354 `#include <wchar.h>`

```
69355 int wscoll(const wchar_t *ws1, const wchar_t *ws2);
69356 CX int wscoll_l(const wchar_t *ws1, const wchar_t *ws2,
69357 locale_t locale);
```

69358 **DESCRIPTION**

69359 CX For `wscoll()`: The functionality described on this reference page is aligned with the ISO C  
 69360 standard. Any conflict between the requirements described here and the ISO C standard is  
 69361 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

69362 CX The `wscoll()` and `wscoll_l()` functions shall compare the wide-character string pointed to by  
 69363 `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the  
 69364 CX `LC_COLLATE` category of the current locale of the process, or the locale represented by `locale`,  
 69365 respectively.

69366 CX The `wscoll()` and `wscoll_l()` functions shall not change the setting of `errno` if successful.

69367 CX An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`  
 69368 or `wscoll_l()`. If `errno` is non-zero on return, an error has occurred.

69369 **RETURN VALUE**

69370 CX Upon successful completion, `wscoll()` and `wscoll_l()` shall return an integer greater than, equal  
 69371 to, or less than 0, according to whether the wide-character string pointed to by `ws1` is greater  
 69372 than, equal to, or less than the wide-character string pointed to by `ws2`, when both are  
 69373 CX interpreted as appropriate to the current locale, or to the locale represented by `locale`,  
 69374 CX respectively. On error, `wscoll()` and `wscoll_l()` shall set `errno`, but no return value is reserved  
 69375 to indicate an error.

69376 **ERRORS**

69377 These functions may fail if:

69378 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of  
 69379 the collating sequence.

69380 The `wscoll_l()` function may fail if:

69381 CX [EINVAL] `locale` is not a valid locale object handle.

69382 **EXAMPLES**

69383 None.

69384 **APPLICATION USAGE**69385 The `wcsxfrm()` and `wscmp()` functions should be used for sorting large lists.69386 **RATIONALE**

69387 None.

69388 **FUTURE DIRECTIONS**

69389 None.

69390 **SEE ALSO**69391 `wscmp()`, `wcsxfrm()`69392 XBD `<wchar.h>`

69393 **CHANGE HISTORY**

69394 First released in Issue 4. Derived from the MSE working draft.

69395 **Issue 5**

69396 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

69397 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

69398 **Issue 7**

69399 The *wscoll\_1()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.  
69400

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcscpy()**69401 **NAME**69402 `wcpcpy`, `wcscpy` — copy a wide-character string, returning a pointer to its end69403 **SYNOPSIS**69404 `#include <wchar.h>`69405 CX `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`  
69406 `wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`69407 **DESCRIPTION**69408 CX For `wcscpy()`: The functionality described on this reference page is aligned with the ISO C  
69409 standard. Any conflict between the requirements described here and the ISO C standard is  
69410 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.69411 CX The `wcpcpy()` and `wcscpy()` functions shall copy the wide-character string pointed to by `ws2`  
69412 (including the terminating null wide-character code) into the array pointed to by `ws1`.69413 The application shall ensure that there is room for at least `wcslen(ws2)+1` wide characters in the  
69414 `ws1` array, and that the `ws2` and `ws1` arrays do not overlap.

69415 If copying takes place between objects that overlap, the behavior is undefined.

69416 **RETURN VALUE**69417 CX The `wcpcpy()` function shall return a pointer to the terminating null wide-character code copied  
69418 into the `ws1` buffer.69419 The `wcscpy()` function shall return `ws1`.

69420 No return values are reserved to indicate an error.

69421 **ERRORS**

69422 No errors are defined.

69423 **EXAMPLES**

69424 None.

69425 **APPLICATION USAGE**

69426 None.

69427 **RATIONALE**

69428 None.

69429 **FUTURE DIRECTIONS**

69430 None.

69431 **SEE ALSO**69432 [strcpy\(\)](#), [wcsdup\(\)](#), [wcsncpy\(\)](#)69433 XBD [<wchar.h>](#)69434 **CHANGE HISTORY**

69435 First released in Issue 4. Derived from the MSE working draft.

69436 **Issue 6**69437 The `wcscpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69438 **Issue 7**69439 The `wcpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API  
69440 Set Part 1.

69441 **NAME**

69442 wscspn — get the length of a complementary wide substring

69443 **SYNOPSIS**

69444 #include &lt;wchar.h&gt;

69445 size\_t wscspn(const wchar\_t \*ws1, const wchar\_t \*ws2);

69446 **DESCRIPTION**

69447 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 69448 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69449 volume of POSIX.1-2008 defers to the ISO C standard.

69450 The *wscspn()* function shall compute the length (in wide characters) of the maximum initial  
 69451 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character  
 69452 codes *not* from the wide-character string pointed to by *ws2*.

69453 **RETURN VALUE**

69454 The *wscspn()* function shall return the length of the initial substring of *ws1*; no return value is  
 69455 reserved to indicate an error.

69456 **ERRORS**

69457 No errors are defined.

69458 **EXAMPLES**

69459 None.

69460 **APPLICATION USAGE**

69461 None.

69462 **RATIONALE**

69463 None.

69464 **FUTURE DIRECTIONS**

69465 None.

69466 **SEE ALSO**69467 [wcssp\(\)](#)

69468 XBD &lt;wchar.h&gt;

69469 **CHANGE HISTORY**

69470 First released in Issue 4. Derived from the MSE working draft.

69471 **Issue 5**

69472 The RETURN VALUE section is updated to indicate that *wscspn()* returns the length of *ws1*,  
 69473 rather than *ws1* itself.

**wcsdup()**69474 **NAME**

69475           wcsdup — duplicate a wide-character string

69476 **SYNOPSIS**

```
69477 CX       #include <wchar.h>
69478       wchar_t *wcsdup(const wchar_t *string);
```

69479 **DESCRIPTION**69480           The *wcsdup()* function is the wide-character equivalent of the *strdup()* function.

69481           The *wcsdup()* function shall return a pointer to a new wide-character string, allocated as if by a  
69482           call to *malloc()*, which is the duplicate of the wide-character string *string*. The returned pointer  
69483           can be passed to *free()*. A null pointer is returned if the new wide-character string cannot be  
69484           created.

69485 **RETURN VALUE**

69486           Upon successful completion, the *wcsdup()* function shall return a pointer to the newly allocated  
69487           wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

69488 **ERRORS**69489           The *wcsdup()* function shall fail if:

69490           [ENOMEM]       Memory large enough for the duplicate string could not be allocated.

69491 **EXAMPLES**

69492           None.

69493 **APPLICATION USAGE**

69494           For functions that allocate memory as if by *malloc()*, the application should release such memory  
69495           when it is no longer required by a call to *free()*. For *wcsdup()*, this is the return value.

69496 **RATIONALE**

69497           None.

69498 **FUTURE DIRECTIONS**

69499           None.

69500 **SEE ALSO**69501           *free()*, *strdup()*, *wscpy()*

69502           XBD &lt;wchar.h&gt;

69503 **CHANGE HISTORY**

69504           First released in Issue 7.

69505 **NAME**69506 `wcsftime` — convert date and time to a wide-character string69507 **SYNOPSIS**69508 `#include <wchar.h>`69509 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`  
69510 `const wchar_t *restrict format, const struct tm *restrict timeptr);`69511 **DESCRIPTION**69512 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
69513 conflict between the requirements described here and the ISO C standard is unintentional. This  
69514 volume of POSIX.1-2008 defers to the ISO C standard.69515 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 69516 • The argument `wcs` points to the initial element of an array of wide characters into which  
69517 the generated output is to be placed.
- 69518 • The argument `maxsize` indicates the maximum number of wide characters to be placed in  
69519 the output array.
- 69520 • The argument `format` is a wide-character string and the conversion specifications are  
69521 replaced by corresponding sequences of wide characters.
- 69522 • The return value indicates the number of wide characters placed in the output array.

69523 If copying takes place between objects that overlap, the behavior is undefined.

69524 **RETURN VALUE**69525 If the total number of resulting wide-character codes including the terminating null wide-  
69526 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character  
69527 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character  
69528 code. Otherwise, zero is returned and the contents of the array are unspecified.69529 **ERRORS**

69530 No errors are defined.

69531 **EXAMPLES**

69532 None.

69533 **APPLICATION USAGE**

69534 None.

69535 **RATIONALE**

69536 None.

69537 **FUTURE DIRECTIONS**

69538 None.

69539 **SEE ALSO**69540 `strftime()`69541 XBD `<wchar.h>`69542 **CHANGE HISTORY**

69543 First released in Issue 4.

**wcsftime()**69544 **Issue 5**

69545 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

69546 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*  
69547 argument is changed from **const char \*** to **const wchar\_t \***.

69548 **Issue 6**

69549 The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69550 **NAME**69551 `wcslen`, `wcsnlen` — get length of a fixed-sized wide-character string69552 **SYNOPSIS**69553 `#include <wchar.h>`69554 `size_t wcslen(const wchar_t *ws);`69555 CX `size_t wcsnlen(const wchar_t *ws, size_t maxlen);`69556 **DESCRIPTION**69557 CX For `wcslen()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.69560 The `wcslen()` function shall compute the number of wide-character codes in the wide-character string to which `ws` points, not including the terminating null wide-character code.69562 CX The `wcsnlen()` function shall compute the smaller of the number of wide characters in the string to which `ws` points, not including the terminating null wide-character code, and the value of `maxlen`. The `wcsnlen()` function shall never examine more than the first `maxlen` characters of the wide-character string pointed to by `ws`.69566 **RETURN VALUE**69567 The `wcslen()` function shall return the length of `ws`.69568 CX The `wcsnlen()` function shall return an integer containing the smaller of either the length of the wide-character string pointed to by `ws` or `maxlen`.

69570 No return values are reserved to indicate an error.

69571 **ERRORS**

69572 No errors are defined.

69573 **EXAMPLES**

69574 None.

69575 **APPLICATION USAGE**

69576 None.

69577 **RATIONALE**

69578 None.

69579 **FUTURE DIRECTIONS**

69580 None.

69581 **SEE ALSO**69582 [strlen\(\)](#)69583 XBD [<wchar.h>](#)69584 **CHANGE HISTORY**

69585 First released in Issue 4. Derived from the MSE working draft.

69586 **Issue 7**69587 The `wcsnlen()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

69588

**wcsncasecmp()**69589 **NAME**69590 `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character string comparison69591 **SYNOPSIS**

```
69592 CX #include <wchar.h>
69593 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
69594 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
69595 size_t n, locale_t locale);
```

69596 **DESCRIPTION**69597 Refer to [wscasecmp\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69598 **NAME**69599 `wcsncat` — concatenate a wide-character string with part of another69600 **SYNOPSIS**69601 `#include <wchar.h>`69602 `wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
69603 `size_t n);`69604 **DESCRIPTION**69605 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
69606 conflict between the requirements described here and the ISO C standard is unintentional. This  
69607 volume of POSIX.1-2008 defers to the ISO C standard.69608 The `wcsncat()` function shall append not more than *n* wide-character codes (a null wide-  
69609 character code and wide-character codes that follow it are not appended) from the array pointed  
69610 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character  
69611 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null  
69612 wide-character code shall always be appended to the result. If copying takes place between  
69613 objects that overlap, the behavior is undefined.69614 **RETURN VALUE**69615 The `wcsncat()` function shall return *ws1*; no return value is reserved to indicate an error.69616 **ERRORS**

69617 No errors are defined.

69618 **EXAMPLES**

69619 None.

69620 **APPLICATION USAGE**

69621 None.

69622 **RATIONALE**

69623 None.

69624 **FUTURE DIRECTIONS**

69625 None.

69626 **SEE ALSO**69627 [wscat\(\)](#)69628 XBD [<wchar.h>](#)69629 **CHANGE HISTORY**

69630 First released in Issue 4. Derived from the MSE working draft.

69631 **Issue 6**69632 The `wcsncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

**wcsncmp()**69633 **NAME**69634 `wcsncmp` — compare part of two wide-character strings69635 **SYNOPSIS**69636 `#include <wchar.h>`69637 `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`69638 **DESCRIPTION**69639 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
69640 conflict between the requirements described here and the ISO C standard is unintentional. This  
69641 volume of POSIX.1-2008 defers to the ISO C standard.69642 The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character  
69643 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*  
69644 to the array pointed to by *ws2*.69645 The sign of a non-zero return value shall be determined by the sign of the difference between the  
69646 values of the first pair of wide-character codes that differ in the objects being compared.69647 **RETURN VALUE**69648 Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less  
69649 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less  
69650 than the possibly null-terminated array pointed to by *ws2*, respectively.69651 **ERRORS**

69652 No errors are defined.

69653 **EXAMPLES**

69654 None.

69655 **APPLICATION USAGE**

69656 None.

69657 **RATIONALE**

69658 None.

69659 **FUTURE DIRECTIONS**

69660 None.

69661 **SEE ALSO**69662 [wcscasecmp\(\)](#), [wcscmp\(\)](#)69663 XBD [<wchar.h>](#)69664 **CHANGE HISTORY**

69665 First released in Issue 4. Derived from the MSE working draft.

69666 **NAME**69667 `wcpncpy`, `wcsncpy` — copy a fixed-size wide-character string, returning a pointer to its end69668 **SYNOPSIS**69669 `#include <wchar.h>`69670 CX `wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,`  
69671 `size_t n);`69672 `wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
69673 `size_t n);`69674 **DESCRIPTION**69675 CX For `wcsncpy()`: The functionality described on this reference page is aligned with the ISO C  
69676 standard. Any conflict between the requirements described here and the ISO C standard is  
69677 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.69678 CX The `wcpncpy()` and `wcsncpy()` functions shall copy not more than *n* wide-character codes (wide-  
69679 character codes that follow a null wide-character code are not copied) from the array pointed to  
69680 by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the  
69681 behavior is undefined.69682 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character  
69683 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,  
69684 until *n* wide-character codes in all are written.69685 **RETURN VALUE**69686 CX If any null wide-character codes were written into the destination, the `wcpncpy()` function shall  
69687 return the address of the first such null wide-character code. Otherwise, it shall return `&ws1[n]`.69688 The `wcsncpy()` function shall return *ws1*.

69689 No return values are reserved to indicate an error.

69690 **ERRORS**

69691 No errors are defined.

69692 **EXAMPLES**

69693 None.

69694 **APPLICATION USAGE**69695 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to  
69696 by *ws2*, the result is not null-terminated.69697 **RATIONALE**

69698 None.

69699 **FUTURE DIRECTIONS**

69700 None.

69701 **SEE ALSO**69702 `strncpy()`, `wcscpy()`69703 XBD `<wchar.h>`69704 **CHANGE HISTORY**

69705 First released in Issue 4. Derived from the MSE working draft.

**wcsncpy()**69706 **Issue 6**

69707 The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69708 **Issue 7**

69709 The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API  
69710 Set Part 1.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wcsnlen()**69711 **NAME**

69712           wcsnlen — get length of a fixed-sized wide-character string

69713 **SYNOPSIS**

69714 CX       #include &lt;wchar.h&gt;

69715       size\_t wcsnlen(const wchar\_t \*ws, size\_t maxlen);

69716 **DESCRIPTION**69717       Refer to *wcslen()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcsnrtombs()***System Interfaces*69718 **NAME**69719 `wcsnrtombs` — convert wide-character string to multi-byte string69720 **SYNOPSIS**69721 CX `#include <wchar.h>`69722 `size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,`  
69723 `size_t nwc, size_t len, mbstate_t *restrict ps);`69724 **DESCRIPTION**69725 Refer to *wcsrtombs()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69726 **NAME**69727 `wcpbrk` — scan a wide-character string for a wide-character code69728 **SYNOPSIS**69729 `#include <wchar.h>`69730 `wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);`69731 **DESCRIPTION**69732 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
69733 conflict between the requirements described here and the ISO C standard is unintentional. This  
69734 volume of POSIX.1-2008 defers to the ISO C standard.69735 The `wcpbrk()` function shall locate the first occurrence in the wide-character string pointed to by  
69736 `ws1` of any wide-character code from the wide-character string pointed to by `ws2`.69737 **RETURN VALUE**69738 Upon successful completion, `wcpbrk()` shall return a pointer to the wide character code or a null  
69739 pointer if no wide-character code from `ws2` occurs in `ws1`.69740 **ERRORS**

69741 No errors are defined.

69742 **EXAMPLES**

69743 None.

69744 **APPLICATION USAGE**

69745 None.

69746 **RATIONALE**

69747 None.

69748 **FUTURE DIRECTIONS**

69749 None.

69750 **SEE ALSO**69751 `wcschr()`, `wcsrchr()`69752 XBD `<wchar.h>`69753 **CHANGE HISTORY**

69754 First released in Issue 4. Derived from the MSE working draft.

**wcsrchr()**69755 **NAME**

69756 wcsrchr — wide-character string scanning operation

69757 **SYNOPSIS**

69758 #include &lt;wchar.h&gt;

69759 wchar\_t \*wcsrchr(const wchar\_t \*ws, wchar\_t wc);

69760 **DESCRIPTION**

69761 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 69762 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69763 volume of POSIX.1-2008 defers to the ISO C standard.

69764 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed  
 69765 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type  
 69766 **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
 69767 terminating null wide-character code shall be considered to be part of the wide-character string.

69768 **RETURN VALUE**

69769 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null  
 69770 pointer if *wc* does not occur in the wide-character string.

69771 **ERRORS**

69772 No errors are defined.

69773 **EXAMPLES**

69774 None.

69775 **APPLICATION USAGE**

69776 None.

69777 **RATIONALE**

69778 None.

69779 **FUTURE DIRECTIONS**

69780 None.

69781 **SEE ALSO**69782 [wcschr\(\)](#)69783 XBD [<wchar.h>](#)69784 **CHANGE HISTORY**

69785 First released in Issue 4. Derived from the MSE working draft.

69786 **Issue 6**

69787 The normative text is updated to avoid use of the term “must” for application requirements.

69788 **NAME**69789 `wcsnrombs, wcsrtombs` — convert a wide-character string to a character string (restartable)69790 **SYNOPSIS**69791 `#include <wchar.h>`

```
69792 CX size_t wcsnrombs(char *restrict dst, const wchar_t **restrict src,  

69793 size_t nwc, size_t len, mbstate_t *restrict ps);  

69794 size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,  

69795 size_t len, mbstate_t *restrict ps);
```

69796 **DESCRIPTION**

69797 CX For `wcsrtombs()`: The functionality described on this reference page is aligned with the ISO C  
69798 standard. Any conflict between the requirements described here and the ISO C standard is  
69799 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

69800 The `wcsrtombs()` function shall convert a sequence of wide characters from the array indirectly  
69801 pointed to by `src` into a sequence of corresponding characters, beginning in the conversion state  
69802 described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters  
69803 shall then be stored into the array pointed to by `dst`. Conversion continues up to and including a  
69804 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the  
69805 following cases:

- 69806 • When a code is reached that does not correspond to a valid character
- 69807 • When the next character would exceed the limit of `len` total bytes to be stored in the array  
69808 pointed to by `dst` (and `dst` is not a null pointer)

69809 Each conversion shall take place as if by a call to the `wcrtomb()` function.

69810 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null  
69811 pointer (if conversion stopped due to reaching a terminating null wide character) or the address  
69812 just past the last wide character converted (if any). If conversion stopped due to reaching a  
69813 terminating null wide character, the resulting state described shall be the initial conversion state.

69814 If `ps` is a null pointer, the `wcsrtombs()` function shall use its own internal `mbstate_t` object, which  
69815 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
69816 pointed to by `ps` shall be used to completely describe the current conversion state of the  
69817 associated character sequence.

69818 CX The `wcsrtombs()` function need not be thread-safe if called with a NULL `ps` argument.

69819 The `wcsnrombs()` function shall be equivalent to the `wcsrtombs()` function, except that the  
69820 conversion is limited to the first `nwc` wide characters.

69821 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

69822 The implementation shall behave as if no function defined in System Interfaces volume of  
69823 POSIX.1-2008 calls these functions.

69824 **RETURN VALUE**

69825 If conversion stops because a code is reached that does not correspond to a valid character, an  
69826 encoding error occurs. In this case, these functions shall store the value of the macro `[EILSEQ]` in  
69827 `errno` and return `(size_t)-1`; the conversion state is undefined. Otherwise, these functions shall  
69828 return the number of bytes in the resulting character sequence, not including the terminating  
69829 null (if any).

**wcsrtombs()**69830 **ERRORS**

69831 These functions shall fail if:

69832 [EILSEQ] A wide-character code does not correspond to a valid character.

69833 These functions may fail if:

69834 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.69835 **EXAMPLES**

69836 None.

69837 **APPLICATION USAGE**

69838 None.

69839 **RATIONALE**

69840 None.

69841 **FUTURE DIRECTIONS**

69842 None.

69843 **SEE ALSO**69844 *mbsinit()*, *wcrtomb()*69845 XBD <*wchar.h*>69846 **CHANGE HISTORY**69847 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
69848 (E).69849 **Issue 6**

69850 In the DESCRIPTION, a note on using this function in a threaded application is added.

69851 Extensions beyond the ISO C standard are marked.

69852 The normative text is updated to avoid use of the term “must” for application requirements.

69853 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69854 **Issue 7**69855 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtombs()* function  
69856 need not be thread-safe if called with a NULL *ps* argument.

69857 Austin Group Interpretation 1003.1-2001 #170 is applied.

69858 The *wcnsrtombs()* function is added from The Open Group Technical Standard, 2006, Extended  
69859 API Set Part 1.

69860 **NAME**69861           wcssp<sub>n</sub> — get the length of a wide substring69862 **SYNOPSIS**

69863           #include &lt;wchar.h&gt;

69864           size\_t wcssp<sub>n</sub>(const wchar\_t \*ws1, const wchar\_t \*ws2);69865 **DESCRIPTION**

69866 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 69867 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69868 volume of POSIX.1-2008 defers to the ISO C standard.

69869       The *wcssp<sub>n</sub>*( ) function shall compute the length (in wide characters) of the maximum initial  
 69870 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character  
 69871 codes from the wide-character string pointed to by *ws2*.

69872 **RETURN VALUE**

69873       The *wcssp<sub>n</sub>*( ) function shall return the length of the initial substring of *ws1*; no return value is  
 69874 reserved to indicate an error.

69875 **ERRORS**

69876       No errors are defined.

69877 **EXAMPLES**

69878       None.

69879 **APPLICATION USAGE**

69880       None.

69881 **RATIONALE**

69882       None.

69883 **FUTURE DIRECTIONS**

69884       None.

69885 **SEE ALSO**69886       *wcscsp<sub>n</sub>*( )

69887       XBD &lt;wchar.h&gt;

69888 **CHANGE HISTORY**

69889       First released in Issue 4. Derived from the MSE working draft.

69890 **Issue 5**

69891       The RETURN VALUE section is updated to indicate that *wcssp<sub>n</sub>*( ) returns the length of *ws1*  
 69892 rather than *ws1* itself.

**wcsstr()**69893 **NAME**69894 `wcsstr` — find a wide-character substring69895 **SYNOPSIS**69896 `#include <wchar.h>`69897 `wchar_t *wcsstr(const wchar_t *restrict ws1,`  
69898 `const wchar_t *restrict ws2);`69899 **DESCRIPTION**69900 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
69901 conflict between the requirements described here and the ISO C standard is unintentional. This  
69902 volume of POSIX.1-2008 defers to the ISO C standard.69903 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by  
69904 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the  
69905 wide-character string pointed to by `ws2`.69906 **RETURN VALUE**69907 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,  
69908 or a null pointer if the wide-character string is not found.69909 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.69910 **ERRORS**

69911 No errors are defined.

69912 **EXAMPLES**

69913 None.

69914 **APPLICATION USAGE**

69915 None.

69916 **RATIONALE**

69917 None.

69918 **FUTURE DIRECTIONS**

69919 None.

69920 **SEE ALSO**69921 [wcschr\(\)](#)69922 XBD [<wchar.h>](#)69923 **CHANGE HISTORY**69924 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
69925 (E).69926 **Issue 6**69927 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69928 **NAME**

69929 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

69930 **SYNOPSIS**

```
69931 #include <wchar.h>
69932 double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
69933 float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
69934 long double wcstold(const wchar_t *restrict nptr,
69935     wchar_t **restrict endptr);
```

69936 **DESCRIPTION**

69937 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 69938 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69939 volume of POSIX.1-2008 defers to the ISO C standard.

69940 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 69941 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the  
 69942 input wide-character string into three parts:

- 69943 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 69944 *iswspace()*)
- 69945 2. A subject sequence interpreted as a floating-point constant or representing infinity or  
 69946 NaN
- 69947 3. A final wide-character string of one or more unrecognized wide-character codes,  
 69948 including the terminating null wide-character code of the input wide-character string

69949 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 69950 the result.

69951 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the  
 69952 following:

- 69953 • A non-empty sequence of decimal digits optionally containing a radix character; then an  
 69954 optional exponent part consisting of the wide character 'e' or the wide character 'E',  
 69955 optionally followed by a '+' or '-' wide character, and then followed by one or more  
 69956 decimal digits
- 69957 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 69958 character; then an optional binary exponent part consisting of the wide character 'p' or  
 69959 the wide character 'P', optionally followed by a '+' or '-' wide character, and then  
 69960 followed by one or more decimal digits
- 69961 • One of INF or INFINITY, or any other wide string equivalent except for case
- 69962 • One of NAN or NAN(*n-wchar-sequence<sub>opt</sub>*), or any other wide string ignoring case in the  
 69963 NAN part, where:

```
69964 n-wchar-sequence:
69965     digit
69966     nondigit
69967     n-wchar-sequence digit
69968     n-wchar-sequence nondigit
```

69969 The subject sequence is defined as the longest initial subsequence of the input wide string,  
 69970 starting with the first non-white-space wide character, that is of the expected form. The subject  
 69971 sequence contains no wide characters if the input wide string is not of the expected form.

**wcstod()**

- 69972 If the subject sequence has the expected form for a floating-point number, the sequence of wide  
 69973 characters starting with the first digit or the radix character (whichever occurs first) shall be  
 69974 interpreted as a floating constant according to the rules of the C language, except that the radix  
 69975 character shall be used in place of a period, and that if neither an exponent part nor a radix  
 69976 character appears in a decimal floating-point number, or if a binary exponent part does not  
 69977 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with  
 69978 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins  
 69979 with a minus-sign, the sequence shall be interpreted as negated. A wide-character sequence INF  
 69980 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it  
 69981 were a floating constant that is too large for the range of the return type. A wide-character  
 69982 sequence NAN or NAN(*n-wchar-sequence<sub>opt</sub>*) shall be interpreted as a quiet NaN, if supported in  
 69983 the return type, else as if it were a subject sequence part that does not have the expected form;  
 69984 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide  
 69985 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.
- 69986 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the  
 69987 conversion shall be rounded in an implementation-defined manner.
- 69988 CX The radix character shall be as defined in the locale of the process (category LC\_NUMERIC). In  
 69989 the POSIX locale, or in a locale where the radix character is not defined, the radix character shall  
 69990 default to a <period> ( ' . ' ).
- 69991 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 69992 accepted.
- 69993 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 69994 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
 69995 *endptr* is not a null pointer.
- 69996 CX The *wcstod()* function shall not change the setting of *errno* if successful.
- 69997 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 69998 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check  
 69999 *errno*.
- 70000 **RETURN VALUE**
- 70001 Upon successful completion, these functions shall return the converted value. If no conversion  
 70002 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].
- 70003 If the correct value is outside the range of representable values, ±HUGE\_VAL, ±HUGE\_VALF, or  
 70004 ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to  
 70005 [ERANGE].
- 70006 If the correct value would cause underflow, a value whose magnitude is no greater than the  
 70007 smallest normalized positive number in the return type shall be returned and *errno* set to  
 70008 [ERANGE].
- 70009 **ERRORS**
- 70010 The *wcstod()* function shall fail if:
- 70011 [ERANGE] The value to be returned would cause overflow or underflow.
- 70012 The *wcstod()* function may fail if:
- 70013 CX [EINVAL] No conversion could be performed.

70014 **EXAMPLES**

70015 None.

70016 **APPLICATION USAGE**

70017 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 70018 result is not exactly representable, the result should be one of the two numbers in the  
 70019 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 70020 extra stipulation that the error should have a correct sign for the current rounding direction.

70021 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in `<float.h>`)  
 70022 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 70023 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 70024 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 70025 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 70026 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 70027 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 70028 correct sign for the current rounding direction.

70029 **RATIONALE**

70030 None.

70031 **FUTURE DIRECTIONS**

70032 None.

70033 **SEE ALSO**70034 [fscanf\(\)](#), [iswspace\(\)](#), [localeconv\(\)](#), [setlocale\(\)](#), [wcstol\(\)](#)70035 XBD Chapter 7 (on page 135), `<float.h>`, `<wchar.h>`70036 **CHANGE HISTORY**

70037 First released in Issue 4. Derived from the MSE working draft.

70038 **Issue 5**70039 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.70040 **Issue 6**

70041 Extensions beyond the ISO C standard are marked.

70042 The following new requirements on POSIX implementations derive from alignment with the  
 70043 Single UNIX Specification:

- 70044 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 70045 added if no conversion could be performed.

70046 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 70047 • The `wcstod()` prototype is updated.
- 70048 • The `wcstof()` and `wcstold()` functions are added.
- 70049 • If the correct value for `wcstod()` would cause underflow, the return value changed from 0  
 70050 (as specified in Issue 5) to the smallest normalized positive number.
- 70051 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are  
 70052 extensively updated.

70053 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

70054 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second  
 70055 paragraph in the RETURN VALUE section.

**wcstod()**

*System Interfaces*

70056 **Issue 7**  
70057

Austin Group Interpretation 1003.1-2001 #015 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70058 **NAME**70059 `wcstoimax`, `wcstoumax` — convert a wide-character string to an integer type70060 **SYNOPSIS**

```
70061 #include <stddef.h>
70062 #include <inttypes.h>
70063
70064 intmax_t wcstoimax(const wchar_t *restrict nptr,
70065                  wchar_t **restrict endptr, int base);
70066 uintmax_t wcstoumax(const wchar_t *restrict nptr,
70067                   wchar_t **restrict endptr, int base);
```

70067 **DESCRIPTION**

70068 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 70069 conflict between the requirements described here and the ISO C standard is unintentional. This  
 70070 volume of POSIX.1-2008 defers to the ISO C standard.

70071 These functions shall be equivalent to the `wcstol()`, `wcstoll()`, `wcstoul()`, and `wcstoull()` functions,  
 70072 respectively, except that the initial portion of the wide string shall be converted to **intmax\_t** and  
 70073 **uintmax\_t** representation, respectively.

70074 **RETURN VALUE**

70075 These functions shall return the converted value, if any.

70076 If no conversion could be performed, zero shall be returned. If the correct value is outside the  
 70077 range of representable values, `{INTMAX_MAX}`, `{INTMAX_MIN}`, or `{UINTMAX_MAX}` shall  
 70078 be returned (according to the return type and sign of the value, if any), and `errno` shall be set to  
 70079 `[ERANGE]`.

70080 **ERRORS**

70081 These functions shall fail if:

- 70082 `[EINVAL]` The value of `base` is not supported.
- 70083 `[ERANGE]` The value to be returned is not representable.

70084 These functions may fail if:

- 70085 `[EINVAL]` No conversion could be performed.

70086 **EXAMPLES**

70087 None.

70088 **APPLICATION USAGE**

70089 None.

70090 **RATIONALE**

70091 None.

70092 **FUTURE DIRECTIONS**

70093 None.

70094 **SEE ALSO**

70095 `wcstol()`, `wcstoul()`

70096 XBD `<inttypes.h>`, `<stddef.h>`

70097 **CHANGE HISTORY**

70098 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**wcstok()**70099 **NAME**70100 `wcstok` — split a wide-character string into tokens70101 **SYNOPSIS**70102 `#include <wchar.h>`70103 `wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
70104 `wchar_t **restrict ptr);`70105 **DESCRIPTION**70106 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70107 conflict between the requirements described here and the ISO C standard is unintentional. This  
70108 volume of POSIX.1-2008 defers to the ISO C standard.70109 A sequence of calls to `wcstok()` shall break the wide-character string pointed to by `ws1` into a  
70110 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-  
70111 character string pointed to by `ws2`. The `ptr` argument points to a caller-provided `wchar_t` pointer  
70112 into which the `wcstok()` function shall store information necessary for it to continue scanning the  
70113 same wide-character string.70114 The first call in the sequence has `ws1` as its first argument, and is followed by calls with a null  
70115 pointer as their first argument. The separator string pointed to by `ws2` may be different from call  
70116 to call.70117 The first call in the sequence shall search the wide-character string pointed to by `ws1` for the first  
70118 wide-character code that is *not* contained in the current separator string pointed to by `ws2`. If no  
70119 such wide-character code is found, then there are no tokens in the wide-character string pointed  
70120 to by `ws1` and `wcstok()` shall return a null pointer. If such a wide-character code is found, it shall  
70121 be the start of the first token.70122 The `wcstok()` function shall then search from there for a wide-character code that *is* contained in  
70123 the current separator string. If no such wide-character code is found, the current token extends  
70124 to the end of the wide-character string pointed to by `ws1`, and subsequent searches for a token  
70125 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a  
70126 null wide character, which terminates the current token. The `wcstok()` function shall save a  
70127 pointer to the following wide-character code, from which the next search for a token shall start.70128 Each subsequent call, with a null pointer as the value of the first argument, shall start searching  
70129 from the saved pointer and behave as described above.70130 The implementation shall behave as if no function calls `wcstok()`.70131 **RETURN VALUE**70132 Upon successful completion, the `wcstok()` function shall return a pointer to the first wide-  
70133 character code of a token. Otherwise, if there is no token, `wcstok()` shall return a null pointer.70134 **ERRORS**

70135 No errors are defined.

70136 **EXAMPLES**

70137 None.

70138 **APPLICATION USAGE**

70139 None.

70140 **RATIONALE**

70141 None.

70142 **FUTURE DIRECTIONS**

70143 None.

70144 **SEE ALSO**70145 XBD [<wchar.h>](#)70146 **CHANGE HISTORY**

70147 First released in Issue 4.

70148 **Issue 5**70149 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is  
70150 added to the definition of *wcstok()* in the SYNOPSIS.70151 **Issue 6**70152 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcstol()**70153 **NAME**70154 `wcstol`, `wcstoll` — convert a wide-character string to a long integer70155 **SYNOPSIS**

```
70156 #include <wchar.h>
70157 long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
70158             int base);
70159 long long wcstoll(const wchar_t *restrict nptr,
70160                  wchar_t **restrict endptr, int base);
```

70161 **DESCRIPTION**

70162 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 70163 conflict between the requirements described here and the ISO C standard is unintentional. This  
 70164 volume of POSIX.1-2008 defers to the ISO C standard.

70165 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 70166 **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

- 70167 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 70168 `iswspace()`)
- 70169 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 70170 value of *base*
- 70171 3. A final wide-character string of one or more unrecognized wide-character codes,  
 70172 including the terminating null wide-character code of the input wide-character string

70173 Then they shall attempt to convert the subject sequence to an integer, and return the result.

70174 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 70175 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 70176 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 70177 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 70178 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 70179 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

70180 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 70181 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 70182 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 70183 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 70184 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code  
 70185 representations of 0x or 0X may optionally precede the sequence of letters and digits, following  
 70186 the sign if present.

70187 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 70188 string, starting with the first non-white-space wide-character code that is of the expected form.  
 70189 The subject sequence contains no wide-character codes if the input wide-character string is  
 70190 empty or consists entirely of white-space wide-character code, or if the first non-white-space  
 70191 wide-character code is other than a sign or a permissible letter or digit.

70192 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
 70193 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
 70194 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for  
 70195 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
 70196 minus-sign, the value resulting from the conversion shall be negated. A pointer to the final  
 70197 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is  
 70198 not a null pointer.

70199 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
70200 accepted.

70201 If the subject sequence is empty or does not have the expected form, no conversion shall be  
70202 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
70203 *endptr* is not a null pointer.

70204 CX These functions shall not change the setting of *errno* if successful.

70205 Since 0, {LONG\_MIN} or {LLONG\_MIN} and {LONG\_MAX} or {LLONG\_MAX} are returned on  
70206 error and are also valid returns on success, an application wishing to check for error situations  
70207 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.

#### 70208 RETURN VALUE

70209 Upon successful completion, these functions shall return the converted value, if any. If no  
70210 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
70211 the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
70212 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
70213 *errno* set to [ERANGE].

#### 70214 ERRORS

70215 These functions shall fail if:

70216 CX [EINVAL] The value of *base* is not supported.

70217 [ERANGE] The value to be returned is not representable.

70218 These functions may fail if:

70219 CX [EINVAL] No conversion could be performed.

#### 70220 EXAMPLES

70221 None.

#### 70222 APPLICATION USAGE

70223 None.

#### 70224 RATIONALE

70225 None.

#### 70226 FUTURE DIRECTIONS

70227 None.

#### 70228 SEE ALSO

70229 *fscanf()*, *iswalpha()*, *wcstod()*

70230 XBD <wchar.h>

#### 70231 CHANGE HISTORY

70232 First released in Issue 4. Derived from the MSE working draft.

#### 70233 Issue 5

70234 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 70235 Issue 6

70236 Extensions beyond the ISO C standard are marked.

**wcstol()**

70237	The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
70238	
70239	<ul style="list-style-type: none"><li>• In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.</li></ul>
70240	
70241	The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
70242	<ul style="list-style-type: none"><li>• The <i>wcstol()</i> prototype is updated.</li><li>• The <i>wcstoll()</i> function is added.</li></ul>
70243	
70244	<b>Issue 7</b>
70245	SD5-XSH-ERN-56 is applied, removing the reference to <b>unsigned long</b> and <b>unsigned long long</b> from the DESCRIPTION.
70246	

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wcstold()**70247 **NAME**70248 `wcstold` — convert a wide-character string to a double-precision number70249 **SYNOPSIS**70250 `#include <wchar.h>`70251 `long double wcstold(const wchar_t *restrict nptr,`70252 `wchar_t **restrict endptr);`70253 **DESCRIPTION**70254 Refer to *wcstod()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcstoll()**70255 **NAME**70256 `wcstoll` — convert a wide-character string to a long integer70257 **SYNOPSIS**70258 `#include <wchar.h>`70259 `long long wcstoll(const wchar_t *restrict nptr,`70260 `wchar_t **restrict endptr, int base);`70261 **DESCRIPTION**70262 Refer to *wcstol()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70263 **NAME**70264 `wcstombs` — convert a wide-character string to a character string70265 **SYNOPSIS**70266 `#include <stdlib.h>`70267 `size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,`  
70268 `size_t n);`70269 **DESCRIPTION**70270 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70271 conflict between the requirements described here and the ISO C standard is unintentional. This  
70272 volume of POSIX.1-2008 defers to the ISO C standard.70273 The `wcstombs()` function shall convert the sequence of wide-character codes that are in the array  
70274 pointed to by `pwcs` into a sequence of characters that begins in the initial shift state and store  
70275 these characters into the array pointed to by `s`, stopping if a character would exceed the limit of `n`  
70276 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to  
70277 `wctomb()`, except that the shift state of `wctomb()` shall not be affected.70278 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.70279 No more than `n` bytes shall be modified in the array pointed to by `s`. If copying takes place  
70280 **CX** between objects that overlap, the behavior is undefined. If `s` is a null pointer, `wcstombs()` shall  
70281 return the length required to convert the entire array regardless of the value of `n`, but no values  
70282 are stored.70283 The `wcstombs()` function need not be thread-safe.70284 **RETURN VALUE**70285 If a wide-character code is encountered that does not correspond to a valid character (of one or  
70286 more bytes each), `wcstombs()` shall return `(size_t)-1`. Otherwise, `wcstombs()` shall return the  
70287 number of bytes stored in the character array, not including any terminating null byte. The array  
70288 shall not be null-terminated if the value returned is `n`.70289 **ERRORS**70290 The `wcstombs()` function shall fail if:70291 **CX** [EILSEQ] A wide-character code does not correspond to a valid character.70292 **EXAMPLES**

70293 None.

70294 **APPLICATION USAGE**

70295 None.

70296 **RATIONALE**

70297 None.

70298 **FUTURE DIRECTIONS**

70299 None.

70300 **SEE ALSO**70301 `mblen()`, `mbtowc()`, `mbstowcs()`, `wctomb()`70302 XBD `<stdlib.h>`

**wcstombs()**70303 **CHANGE HISTORY**

70304 First released in Issue 4. Derived from the ISO C standard.

70305 **Issue 6**

70306 The following new requirements on POSIX implementations derive from alignment with the  
70307 Single UNIX Specification:

- 70308 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 70309 • The [EILSEQ] error condition is added.

70310 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

70311 **Issue 7**

70312 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70313 **NAME**70314 `wcstoul`, `wcstoull` — convert a wide-character string to an unsigned long70315 **SYNOPSIS**

```
70316 #include <wchar.h>
70317 unsigned long wcstoul(const wchar_t *restrict nptr,
70318     wchar_t **restrict endptr, int base);
70319 unsigned long long wcstoull(const wchar_t *restrict nptr,
70320     wchar_t **restrict endptr, int base);
```

70321 **DESCRIPTION**

70322 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 70323 conflict between the requirements described here and the ISO C standard is unintentional. This  
 70324 volume of POSIX.1-2008 defers to the ISO C standard.

70325 The `wcstoul()` and `wcstoull()` functions shall convert the initial portion of the wide-character  
 70326 string pointed to by `nptr` to **unsigned long** and **unsigned long long** representation, respectively.  
 70327 First, they shall decompose the input wide-character string into three parts:

- 70328 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 70329 `iswspace()`)
- 70330 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 70331 value of `base`
- 70332 3. A final wide-character string of one or more unrecognized wide-character codes,  
 70333 including the terminating null wide-character code of the input wide-character string

70334 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
 70335 result.

70336 If `base` is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 70337 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 70338 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 70339 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 70340 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 70341 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

70342 If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence  
 70343 of letters and digits representing an integer with the radix specified by `base`, optionally preceded  
 70344 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 70345 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 70346 than that of `base` shall be permitted. If the value of `base` is 16, the wide-character codes 0x or 0X  
 70347 may optionally precede the sequence of letters and digits, following the sign if present.

70348 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 70349 string, starting with the first wide-character code that is not white space and is of the expected  
 70350 form. The subject sequence contains no wide-character codes if the input wide-character string is  
 70351 empty or consists entirely of white-space wide-character codes, or if the first wide-character  
 70352 code that is not white space is other than a sign or a permissible letter or digit.

70353 If the subject sequence has the expected form and `base` is 0, the sequence of wide-character codes  
 70354 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
 70355 the expected form and the value of `base` is between 2 and 36, it shall be used as the base for  
 70356 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
 70357 minus-sign, the value resulting from the conversion shall be negated. A pointer to the final  
 70358 wide-character string shall be stored in the object pointed to by `endptr`, provided that `endptr` is

**wcstoul()**

- 70359 not a null pointer.
- 70360 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
70361 accepted.
- 70362 If the subject sequence is empty or does not have the expected form, no conversion shall be  
70363 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
70364 *endptr* is not a null pointer.
- 70365 CX The *wcstoul()* function shall not change the setting of *errno* if successful.
- 70366 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and 0 is also a valid return  
70367 on success, an application wishing to check for error situations should set *errno* to 0, then call  
70368 *wcstoul()* or *wcstoull()*, then check *errno*.
- 70369 **RETURN VALUE**
- 70370 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted  
70371 CX value, if any. If no conversion could be performed, 0 shall be returned, and *errno* may be set to  
70372 indicate the error. If the correct value is outside the range of representable values,  
70373 {ULONG\_MAX} or {ULLONG\_MAX} respectively shall be returned and *errno* set to [ERANGE].
- 70374 **ERRORS**
- 70375 These functions shall fail if:
- 70376 CX [EINVAL] The value of *base* is not supported.
- 70377 [ERANGE] The value to be returned is not representable.
- 70378 These functions may fail if:
- 70379 CX [EINVAL] No conversion could be performed.
- 70380 **EXAMPLES**
- 70381 None.
- 70382 **APPLICATION USAGE**
- 70383 None.
- 70384 **RATIONALE**
- 70385 None.
- 70386 **FUTURE DIRECTIONS**
- 70387 None.
- 70388 **SEE ALSO**
- 70389 *fscanf()*, *iswalpha()*, *wcstod()*, *wcstol()*
- 70390 XBD <wchar.h>
- 70391 **CHANGE HISTORY**
- 70392 First released in Issue 4. Derived from the MSE working draft.
- 70393 **Issue 5**
- 70394 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 70395 **Issue 6**
- 70396 Extensions beyond the ISO C standard are marked.

70397 The following new requirements on POSIX implementations derive from alignment with the  
70398 Single UNIX Specification:

- 70399 • The [EINVAL] error condition is added for when the value of *base* is not supported.

70400 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
70401 added if no conversion could be performed.

70402 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 70403 • The *wcstoul()* prototype is updated.
- 70404 • The *wcstoull()* function is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wcstoumax()**70405 **NAME**70406 `wcstoumax` — convert a wide-character string to an integer type70407 **SYNOPSIS**70408 `#include <stddef.h>`70409 `#include <inttypes.h>`70410 `uintmax_t wcstoumax(const wchar_t *restrict nptr,`70411 `wchar_t **restrict endptr, int base);`70412 **DESCRIPTION**70413 Refer to *wcstoimax()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70414 **NAME**

70415           wcswidth — number of column positions of a wide-character string

70416 **SYNOPSIS**

```
70417 XSI      #include <wchar.h>
70418      int wcswidth(const wchar_t *pwcs, size_t n);
```

70419 **DESCRIPTION**

70420       The *wcswidth()* function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

70423 **RETURN VALUE**

70424       The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

70428 **ERRORS**

70429       No errors are defined.

70430 **EXAMPLES**

70431       None.

70432 **APPLICATION USAGE**

70433       This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

70435 **RATIONALE**

70436       None.

70437 **FUTURE DIRECTIONS**

70438       None.

70439 **SEE ALSO**70440       *wcwidth()*

70441       XBD Section 3.103 (on page 50), &lt;wchar.h&gt;

70442 **CHANGE HISTORY**

70443       First released in Issue 4. Derived from the MSE working draft.

70444 **Issue 6**

70445       The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

**wcsxfrm()**70446 **NAME**70447 `wcsxfrm, wcsxfrm_l` — wide-character string transformation70448 **SYNOPSIS**70449 `#include <wchar.h>`70450 `size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
70451 `size_t n);`70452 CX `size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
70453 `size_t n, locale_t locale);`70454 **DESCRIPTION**70455 CX For `wcsxfrm()`: The functionality described on this reference page is aligned with the ISO C  
70456 standard. Any conflict between the requirements described here and the ISO C standard is  
70457 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.70458 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall transform the wide-character string pointed to  
70459 by `ws2` and place the resulting wide-character string into the array pointed to by `ws1`. The  
70460 transformation shall be such that if `wcscmp()` is applied to two transformed wide strings, it shall  
70461 CX return a value greater than, equal to, or less than 0, corresponding to the result of `wcscoll()` and  
70462 `wcscoll_l()` applied to the same two original wide-character strings, and the same `LC_COLLATE`  
70463 CX category of the locale of the process or the locale object `locale`, respectively. No more than `n`  
70464 wide-character codes shall be placed into the resulting array pointed to by `ws1`, including the  
70465 terminating null wide-character code. If `n` is 0, `ws1` is permitted to be a null pointer. If copying  
70466 takes place between objects that overlap, the behavior is undefined.70467 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall not change the setting of `errno` if successful.70468 Since no return value is reserved to indicate an error, an application wishing to check for error  
70469 situations should set `errno` to 0, then call `wcsxfrm()` or `wcsxfrm_l()`, then check `errno`.70470 **RETURN VALUE**70471 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall return the length of the transformed wide-  
70472 character string (not including the terminating null wide-character code). If the value returned is  
70473 `n` or more, the contents of the array pointed to by `ws1` are unspecified.70474 CX On error, the `wcsxfrm()` and `wcsxfrm_l()` functions may set `errno`, but no return value is reserved  
70475 to indicate an error.70476 **ERRORS**

70477 These functions may fail if:

70478 CX [EINVAL] The wide-character string pointed to by `ws2` contains wide-character codes  
70479 outside the domain of the collating sequence.70480 The `wcsxfrm_l()` function may fail if:70481 CX [EINVAL] `locale` is not a valid locale object handle.

70482 **EXAMPLES**

70483 None.

70484 **APPLICATION USAGE**

70485 The transformation function is such that two transformed wide-character strings can be ordered  
 70486 by `wscmp()` as appropriate to collating sequence information in the locale of the process  
 70487 (category `LC_COLLATE`).

70488 The fact that when  $n$  is 0 `ws1` is permitted to be a null pointer is useful to determine the size of  
 70489 the `ws1` array prior to making the transformation.

70490 **RATIONALE**

70491 None.

70492 **FUTURE DIRECTIONS**

70493 None.

70494 **SEE ALSO**70495 `wscmp()`, `wscoll()`70496 XBD `<wchar.h>`70497 **CHANGE HISTORY**

70498 First released in Issue 4. Derived from the MSE working draft.

70499 **Issue 5**

70500 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

70501 The DESCRIPTION is updated to indicate that `errno` is not changed if the function is successful.70502 **Issue 6**70503 In earlier versions, this function was required to return `-1` on error.

70504 Extensions beyond the ISO C standard are marked.

70505 The following new requirements on POSIX implementations derive from alignment with the  
 70506 Single UNIX Specification:

- 70507 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 70508 added if no conversion could be performed.

70509 The `wcsxfrm()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.70510 **Issue 7**

70511 The `wcsxfrm_l()` function is added from The Open Group Technical Standard, 2006, Extended  
 70512 API Set Part 4.

**wctob()**70513 **NAME**70514 `wctob` — wide-character to single-byte conversion70515 **SYNOPSIS**

```
70516 #include <stdio.h>
70517 #include <wchar.h>
70518 int wctob(wint_t c);
```

70519 **DESCRIPTION**

70520 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 70521 conflict between the requirements described here and the ISO C standard is unintentional. This  
 70522 volume of POSIX.1-2008 defers to the ISO C standard.

70523 The `wctob()` function shall determine whether `c` corresponds to a member of the extended  
 70524 character set whose character representation is a single byte when in the initial shift state.

70525 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

70526 **RETURN VALUE**

70527 The `wctob()` function shall return EOF if `c` does not correspond to a character with length one in  
 70528 the initial shift state. Otherwise, it shall return the single-byte representation of that character as  
 70529 an **unsigned char** converted to **int**.

70530 **ERRORS**

70531 No errors are defined.

70532 **EXAMPLES**

70533 None.

70534 **APPLICATION USAGE**

70535 None.

70536 **RATIONALE**

70537 None.

70538 **FUTURE DIRECTIONS**

70539 None.

70540 **SEE ALSO**

70541 [btowc\(\)](#)

70542 XBD [<stdio.h>](#) [<wchar.h>](#)

70543 **CHANGE HISTORY**

70544 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 70545 (E).

70546 **NAME**70547 `wctomb` — convert a wide-character code to a character70548 **SYNOPSIS**70549 `#include <stdlib.h>`70550 `int wctomb(char *s, wchar_t wchar);`70551 **DESCRIPTION**

70552 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 70553 conflict between the requirements described here and the ISO C standard is unintentional. This  
 70554 volume of POSIX.1-2008 defers to the ISO C standard.

70555 The `wctomb()` function shall determine the number of bytes needed to represent the character  
 70556 corresponding to the wide-character code whose value is `wchar` (including any change in the  
 70557 shift state). It shall store the character representation (possibly multiple bytes and any special  
 70558 bytes to change shift state) in the array object pointed to by `s` (if `s` is not a null pointer). At most  
 70559 `{MB_CUR_MAX}` bytes shall be stored. If `wchar` is 0, a null byte shall be stored, preceded by any  
 70560 shift sequence needed to restore the initial shift state, and `wctomb()` shall be left in the initial shift  
 70561 state.

70562 CX The behavior of this function is affected by the `LC_CTYPE` category of the current locale. For a  
 70563 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
 70564 character pointer argument, `s`, is a null pointer. Subsequent calls with `s` as other than a null  
 70565 pointer shall cause the internal state of the function to be altered as necessary. A call with `s` as a  
 70566 null pointer shall cause this function to return a non-zero value if encodings have state  
 70567 dependency, and 0 otherwise. Changing the `LC_CTYPE` category causes the shift state of this  
 70568 function to be unspecified.

70569 The `wctomb()` function need not be thread-safe.

70570 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls  
 70571 `wctomb()`.

70572 **RETURN VALUE**

70573 If `s` is a null pointer, `wctomb()` shall return a non-zero or 0 value, if character encodings,  
 70574 respectively, do or do not have state-dependent encodings. If `s` is not a null pointer, `wctomb()`  
 70575 shall return `-1` if the value of `wchar` does not correspond to a valid character, or return the  
 70576 number of bytes that constitute the character corresponding to the value of `wchar`.

70577 In no case shall the value returned be greater than the value of the `{MB_CUR_MAX}` macro.

70578 **ERRORS**

70579 The `wctomb()` function shall fail if:

70580 CX **[EILSEQ]** An invalid wide-character code is detected.

70581 **EXAMPLES**

70582 None.

70583 **APPLICATION USAGE**

70584 None.

70585 **RATIONALE**

70586 None.

**wctomb()**70587 **FUTURE DIRECTIONS**

70588 None.

70589 **SEE ALSO**70590 *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*70591 XBD [<stdlib.h>](#)70592 **CHANGE HISTORY**

70593 First released in Issue 4. Derived from the ANSI C standard.

70594 **Issue 6**

70595 Extensions beyond the ISO C standard are marked.

70596 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

70597 **Issue 7**

70598 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70599 **NAME**70600 `wctrans, wctrans_l` — define character mapping70601 **SYNOPSIS**70602 `#include <wctype.h>`70603 `wctrans_t wctrans(const char *charclass);`70604 CX `wctrans_t wctrans_l(const char *charclass, locale_t locale);`70605 **DESCRIPTION**70606 CX For `wctrans()`: The functionality described on this reference page is aligned with the ISO C  
70607 standard. Any conflict between the requirements described here and the ISO C standard is  
70608 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.70609 CX The `wctrans()` and `wctrans_l()` functions are defined for valid character mapping names  
70610 identified in the current locale. The `charclass` is a string identifying a generic character mapping  
70611 name for which codeset-specific information is required. The following character mapping  
70612 names are defined in all locales: **tolower** and **toupper**.70613 These functions shall return a value of type **wctrans\_t**, which can be used as the second  
70614 CX argument to subsequent calls of `towctrans()` and `towctrans_l()`.70615 CX The `wctrans()` and `wctrans_l()` functions shall determine values of **wctrans\_t** according to the  
70616 rules of the coded character set defined by character mapping information in the locale of the  
70617 CX process or in the locale represented by `locale`, respectively (category `LC_CTYPE`).70618 The values returned by `wctrans()` shall be valid until a call to `setlocale()` that modifies the  
70619 category `LC_CTYPE`.70620 CX The values returned by `wctrans_l()` shall be valid only in calls to `wctrans_l()` with a locale  
70621 represented by `locale` with the same `LC_CTYPE` category value.70622 **RETURN VALUE**70623 CX The `wctrans()` and `wctrans_l()` functions shall return 0 and may set `errno` to indicate the error if  
70624 the given character mapping name is not valid for the current locale (category `LC_CTYPE`);  
70625 otherwise, they shall return a non-zero object of type **wctrans\_t** that can be used in calls to  
70626 CX `towctrans()` and `towctrans_l()`.70627 **ERRORS**

70628 These functions may fail if:

70629 CX **[EINVAL]** The character mapping name pointed to by `charclass` is not valid in the current  
70630 locale.70631 The `wctrans_l()` function may fail if:70632 CX **[EINVAL]** `locale` is not a valid locale object handle.70633 **EXAMPLES**

70634 None.

70635 **APPLICATION USAGE**

70636 None.

70637 **RATIONALE**

70638 None.

**wctrans()**

System Interfaces

70639 **FUTURE DIRECTIONS**

70640 None.

70641 **SEE ALSO**70642 [towctrans\(\)](#)70643 XBD [<wctype.h>](#)70644 **CHANGE HISTORY**

70645 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

70646 **Issue 7**70647 The *wctrans\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
70648 API Set Part 4.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

70649 **NAME**

70650 wctype, wctype\_l — define character class

70651 **SYNOPSIS**

70652 #include &lt;wctype.h&gt;

70653 wctype\_t wctype(const char \*property);

70654 CX wctype\_t wctype\_l(const char \*property, locale\_t locale);

70655 **DESCRIPTION**70656 CX For *wctype()*: The functionality described on this reference page is aligned with the ISO C  
70657 standard. Any conflict between the requirements described here and the ISO C standard is  
70658 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.70659 CX The *wctype()* and *wctype\_l()* functions are defined for valid character class names as defined in  
70660 CX the current locale or in the locale represented by *locale*, respectively.70661 The *property* argument is a string identifying a generic character class for which codeset-specific  
70662 type information is required. The following character class names shall be defined in all locales:

70663	<b>alnum</b>	<b>digit</b>	<b>punct</b>
70664	<b>alpha</b>	<b>graph</b>	<b>space</b>
70665	<b>blank</b>	<b>lower</b>	<b>upper</b>
70666	<b>cntrl</b>	<b>print</b>	<b>xdigit</b>

70667 Additional character class names defined in the locale definition file (category *LC\_CTYPE*) can  
70668 also be specified.70669 These functions shall return a value of type **wctype\_t**, which can be used as the second  
70670 CX argument to subsequent calls of *iswctype()* and *iswctype\_l()*.70671 CX The *wctype()* and *wctype\_l()* functions shall determine values of **wctype\_t** according to the  
70672 rules of the coded character set defined by character type information in the locale of the process  
70673 CX or in the locale represented by *locale*, respectively (category *LC\_CTYPE*).70674 The values returned by *wctype()* shall be valid until a call to *setlocale()* that modifies the category  
70675 *LC\_CTYPE*.70676 CX The values returned by *wctype\_l()* shall be valid only in calls to *iswctype\_l()* with a locale  
70677 represented by *locale* with the same *LC\_CTYPE* category value.70678 **RETURN VALUE**70679 CX The *wctype()* and *wctype\_l()* functions shall return 0 if the given character class name is not  
70680 valid for the current locale (category *LC\_CTYPE*); otherwise, they shall return an object of type70681 CX **wctype\_t** that can be used in calls to *iswctype()* and *iswctype\_l()*.70682 **ERRORS**70683 The *wctype\_l()* function may fail if:70684 CX [EINVAL] *locale* is not a valid locale object handle.

**wctype()**70685 **EXAMPLES**

70686 None.

70687 **APPLICATION USAGE**

70688 None.

70689 **RATIONALE**

70690 None.

70691 **FUTURE DIRECTIONS**

70692 None.

70693 **SEE ALSO**70694 [iswctype\(\)](#)70695 XBD [<wctype.h>](#)70696 **CHANGE HISTORY**

70697 First released in Issue 4.

70698 **Issue 5**70699 The following change has been made in this version for alignment with  
70700 ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- 70701
- The SYNOPSIS has been changed to indicate that this function and associated data types  
70702 are now made visible by inclusion of the [<wctype.h>](#) header rather than [<wchar.h>](#).

70703 **Issue 7**70704 The [wctype\\_l\(\)](#) function is added from The Open Group Technical Standard, 2006, Extended API  
70705 Set Part 4.

70706 **NAME**70707 `wcwidth` — number of column positions of a wide-character code70708 **SYNOPSIS**

```
70709 xSI #include <wchar.h>
70710 int wcwidth(wchar_t wc);
```

70711 **DESCRIPTION**

70712 The `wcwidth()` function shall determine the number of column positions required for the wide  
 70713 character `wc`. The application shall ensure that the value of `wc` is a character representable as a  
 70714 **wchar\_t**, and is a wide-character code corresponding to a valid character in the current locale.

70715 **RETURN VALUE**

70716 The `wcwidth()` function shall either return 0 (if `wc` is a null wide-character code), or return the  
 70717 number of column positions to be occupied by the wide-character code `wc`, or return -1 (if `wc`  
 70718 does not correspond to a printable wide-character code).

70719 **ERRORS**

70720 No errors are defined.

70721 **EXAMPLES**

70722 None.

70723 **APPLICATION USAGE**

70724 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
 70725 return value for a non-printable wide character is not specified.

70726 **RATIONALE**

70727 None.

70728 **FUTURE DIRECTIONS**

70729 None.

70730 **SEE ALSO**70731 [wcswidth\(\)](#)70732 XBD [<wchar.h>](#)70733 **CHANGE HISTORY**

70734 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
 70735 draft.

70736 **Issue 6**

70737 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

70738 The normative text is updated to avoid use of the term “must” for application requirements.

**wmemchr()**70739 **NAME**70740 `wmemchr` — find a wide character in memory70741 **SYNOPSIS**70742 `#include <wchar.h>`70743 `wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);`70744 **DESCRIPTION**70745 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70746 conflict between the requirements described here and the ISO C standard is unintentional. This  
70747 volume of POSIX.1-2008 defers to the ISO C standard.70748 The `wmemchr()` function shall locate the first occurrence of `wc` in the initial `n` wide characters of  
70749 the object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values  
70750 shall be treated identically. The null wide character and `wchar_t` values not corresponding to  
70751 valid characters shall not be treated specially.70752 If `n` is zero, the application shall ensure that `ws` is a valid pointer and the function behaves as if  
70753 no valid occurrence of `wc` is found.70754 **RETURN VALUE**70755 The `wmemchr()` function shall return a pointer to the located wide character, or a null pointer if  
70756 the wide character does not occur in the object.70757 **ERRORS**

70758 No errors are defined.

70759 **EXAMPLES**

70760 None.

70761 **APPLICATION USAGE**

70762 None.

70763 **RATIONALE**

70764 None.

70765 **FUTURE DIRECTIONS**

70766 None.

70767 **SEE ALSO**70768 `wmemcmp()`, `wmemcpy()`, `wmemmove()`, `wmemset()`70769 XBD `<wchar.h>`70770 **CHANGE HISTORY**70771 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
70772 (E).70773 **Issue 6**

70774 The normative text is updated to avoid use of the term “must” for application requirements.

70775 **NAME**

70776           wmemcmp — compare wide characters in memory

70777 **SYNOPSIS**

70778           #include &lt;wchar.h&gt;

70779           int wmemcmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

70780 **DESCRIPTION**70781 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
70782 conflict between the requirements described here and the ISO C standard is unintentional. This  
70783 volume of POSIX.1-2008 defers to the ISO C standard.70784       The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by  
70785 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be  
70786 affected by locale and all **wchar\_t** values shall be treated identically. The null wide character and  
70787 **wchar\_t** values not corresponding to valid characters shall not be treated specially.70788       If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
70789 shall behave as if the two objects compare equal.70790 **RETURN VALUE**70791       The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,  
70792 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object  
70793 pointed to by *ws2*.70794 **ERRORS**

70795       No errors are defined.

70796 **EXAMPLES**

70797       None.

70798 **APPLICATION USAGE**

70799       None.

70800 **RATIONALE**

70801       None.

70802 **FUTURE DIRECTIONS**

70803       None.

70804 **SEE ALSO**70805       *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*

70806       XBD &lt;wchar.h&gt;

70807 **CHANGE HISTORY**70808       First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
70809 (E).70810 **Issue 6**

70811       The normative text is updated to avoid use of the term “must” for application requirements.

**wmemcpy()**70812 **NAME**70813 `wmemcpy` — copy wide characters in memory70814 **SYNOPSIS**70815 `#include <wchar.h>`70816 `wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
70817 `size_t n);`70818 **DESCRIPTION**70819 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70820 conflict between the requirements described here and the ISO C standard is unintentional. This  
70821 volume of POSIX.1-2008 defers to the ISO C standard.70822 The `wmemcpy()` function shall copy *n* wide characters from the object pointed to by *ws2* to the  
70823 object pointed to by *ws1*. This function shall not be affected by locale and all `wchar_t` values  
70824 shall be treated identically. The null wide character and `wchar_t` values not corresponding to  
70825 valid characters shall not be treated specially.70826 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
70827 shall copy zero wide characters.70828 **RETURN VALUE**70829 The `wmemcpy()` function shall return the value of *ws1*.70830 **ERRORS**

70831 No errors are defined.

70832 **EXAMPLES**

70833 None.

70834 **APPLICATION USAGE**

70835 None.

70836 **RATIONALE**

70837 None.

70838 **FUTURE DIRECTIONS**

70839 None.

70840 **SEE ALSO**70841 `wmemchr()`, `wmemcmp()`, `wmemmove()`, `wmemset()`70842 XBD `<wchar.h>`70843 **CHANGE HISTORY**70844 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
70845 (E).70846 **Issue 6**

70847 The normative text is updated to avoid use of the term “must” for application requirements.

70848 The `wmemcpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

70849 **NAME**70850 `wmemmove` — copy wide characters in memory with overlapping areas70851 **SYNOPSIS**70852 `#include <wchar.h>`70853 `wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);`70854 **DESCRIPTION**70855 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70856 conflict between the requirements described here and the ISO C standard is unintentional. This  
70857 volume of POSIX.1-2008 defers to the ISO C standard.70858 The `wmemmove()` function shall copy  $n$  wide characters from the object pointed to by `ws2` to the  
70859 object pointed to by `ws1`. Copying shall take place as if the  $n$  wide characters from the object  
70860 pointed to by `ws2` are first copied into a temporary array of  $n$  wide characters that does not  
70861 overlap the objects pointed to by `ws1` or `ws2`, and then the  $n$  wide characters from the temporary  
70862 array are copied into the object pointed to by `ws1`.70863 This function shall not be affected by locale and all `wchar_t` values shall be treated identically.  
70864 The null wide character and `wchar_t` values not corresponding to valid characters shall not be  
70865 treated specially.70866 If  $n$  is zero, the application shall ensure that `ws1` and `ws2` are valid pointers, and the function  
70867 shall copy zero wide characters.70868 **RETURN VALUE**70869 The `wmemmove()` function shall return the value of `ws1`.70870 **ERRORS**

70871 No errors are defined

70872 **EXAMPLES**

70873 None.

70874 **APPLICATION USAGE**

70875 None.

70876 **RATIONALE**

70877 None.

70878 **FUTURE DIRECTIONS**

70879 None.

70880 **SEE ALSO**70881 `wmemcpy()`, `wmemcmp()`, `wmemcpy()`, `wmemset()`70882 XBD `<wchar.h>`70883 **CHANGE HISTORY**70884 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
70885 (E).70886 **Issue 6**

70887 The normative text is updated to avoid use of the term “must” for application requirements.

**wmemset()**70888 **NAME**70889 `wmemset` — set wide characters in memory70890 **SYNOPSIS**70891 `#include <wchar.h>`70892 `wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);`70893 **DESCRIPTION**70894 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
70895 conflict between the requirements described here and the ISO C standard is unintentional. This  
70896 volume of POSIX.1-2008 defers to the ISO C standard.70897 The `wmemset()` function shall copy the value of `wc` into each of the first `n` wide characters of the  
70898 object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values shall  
70899 be treated identically. The null wide character and `wchar_t` values not corresponding to valid  
70900 characters shall not be treated specially.70901 If `n` is zero, the application shall ensure that `ws` is a valid pointer, and the function shall copy  
70902 zero wide characters.70903 **RETURN VALUE**70904 The `wmemset()` functions shall return the value of `ws`.70905 **ERRORS**

70906 No errors are defined.

70907 **EXAMPLES**

70908 None.

70909 **APPLICATION USAGE**

70910 None.

70911 **RATIONALE**

70912 None.

70913 **FUTURE DIRECTIONS**

70914 None.

70915 **SEE ALSO**70916 `wmemchr()`, `wmemcmp()`, `wmemcpy()`, `wmemmove()`70917 XBD `<wchar.h>`70918 **CHANGE HISTORY**70919 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
70920 (E).70921 **Issue 6**

70922 The normative text is updated to avoid use of the term “must” for application requirements.

70923 **NAME**

70924 wordexp, wordfree — perform word expansions

70925 **SYNOPSIS**

70926 #include &lt;wordexp.h&gt;

70927 int wordexp(const char \*restrict words, wordexp\_t \*restrict pwordexp,  
70928 int flags);

70929 void wordfree(wordexp\_t \*pwordexp);

70930 **DESCRIPTION**70931 The *wordexp()* function shall perform word expansions as described in XCU Section 2.6 (on page  
70932 2305), subject to quoting as described in XCU Section 2.2 (on page 2298), and place the list of  
70933 expanded words into the structure pointed to by *pwordexp*.70934 The *words* argument is a pointer to a string containing one or more words to be expanded. The  
70935 expansions shall be the same as would be performed by the command line interpreter if *words*  
70936 were the part of a command line representing the arguments to a utility. Therefore, the  
70937 application shall ensure that *words* does not contain an unquoted <newline> character or any of  
70938 the unquoted shell special characters ' | ', ' & ', ' ; ', ' < ', ' > ' except in the context of command  
70939 substitution as specified in XCU Section 2.6.3 (on page 2309). It also shall not contain unquoted  
70940 parentheses or braces, except in the context of command or variable substitution. The  
70941 application shall ensure that every member of *words* which it expects to have expanded by  
70942 *wordexp()* does not contain an unquoted initial comment character. The application shall also  
70943 ensure that any words which it intends to be ignored (because they begin or continue a  
70944 comment) are deleted from *words*. If the argument *words* contains an unquoted comment  
70945 character (<number-sign>) that is the beginning of a token, *wordexp()* shall either treat the  
70946 comment character as a regular character, or interpret it as a comment indicator and ignore the  
70947 remainder of *words*.70948 The structure type **wordexp\_t** is defined in the <wordexp.h> header and includes at least the  
70949 following members:

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp-&gt;we_wordv</i> .

70955 The *wordexp()* function shall store the number of generated words into *pwordexp->we\_wordc* and  
70956 a pointer to a list of pointers to words in *pwordexp->we\_wordv*. Each individual field created  
70957 during field splitting (see XCU Section 2.6.5, on page 2311) or pathname expansion (see XCU  
70958 Section 2.6.6, on page 2311) shall be a separate word in the *pwordexp->we\_wordv* list. The words  
70959 shall be in order as described in XCU Section 2.6 (on page 2305). The first pointer after the last  
70960 word pointer shall be a null pointer. The expansion of special parameters described in XCU  
70961 Section 2.5.2 (on page 2302) is unspecified.70962 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*  
70963 function shall allocate other space as needed, including memory pointed to by  
70964 *pwordexp->we\_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a  
70965 previous call to *wordexp()*.70966 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the bitwise-  
70967 inclusive OR of zero or more of the following constants, which are defined in <wordexp.h>:

**wordexp()**

70968	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
70969	WRDE_DOOFFS	Make use of <i>pwordexp-&gt;we_offs</i> . If this flag is set, <i>pwordexp-&gt;we_offs</i> is used to specify how many null pointers to add to the beginning of <i>pwordexp-&gt;we_wordv</i> . In other words, <i>pwordexp-&gt;we_wordv</i> shall point to <i>pwordexp-&gt;we_offs</i> null pointers, followed by <i>pwordexp-&gt;we_wordc</i> word pointers, followed by a null pointer.
70970		
70971		
70972		
70973		
70974	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2008, fail if command substitution, as specified in XCU Section 2.6.3 (on page 2309), is requested.
70975		
70976		
70977	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
70978		
70979		
70980		
70981	WRDE_SHOWERR	Do not redirect <i>stderr</i> to <i>/dev/null</i> .
70982	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
70983	The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to <i>wordexp()</i> . The following rules apply to applications when two or more calls to <i>wordexp()</i> are made with the same value of <i>pwordexp</i> and without intervening calls to <i>wordfree()</i> :	
70984		
70985		
70986	1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.	
70987	2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.	
70988	3. After the second and each subsequent call, <i>pwordexp-&gt;we_wordv</i> shall point to a list containing the following:	
70989		
70990	a. Zero or more null pointers, as specified by WRDE_DOOFFS and	
70991	<i>pwordexp-&gt;we_offs</i>	
70992	b. Pointers to the words that were in the <i>pwordexp-&gt;we_wordv</i> list before the call, in	
70993	the same order as before	
70994	c. Pointers to the new words generated by the latest call, in the specified order	
70995	4. The count returned in <i>pwordexp-&gt;we_wordc</i> shall be the total number of words from all of	
70996	the calls.	
70997	5. The application can change any of the fields after a call to <i>wordexp()</i> , but if it does it shall	
70998	reset them to the original value before a subsequent call, using the same <i>pwordexp</i> value,	
70999	to <i>wordfree()</i> or <i>wordexp()</i> with the WRDE_APPEND or WRDE_REUSE flag.	
71000	If the implementation supports the utilities defined in the Shell and Utilities volume of	
71001	POSIX.1-2008, and <i>words</i> contains an unquoted character—<newline>, '   ', ' & ', ' ; ', ' < ', ' > ',	
71002	' ( ', ' ) ', ' { ', ' } '—in an inappropriate context, <i>wordexp()</i> shall fail, and the number of	
71003	expanded words shall be 0.	
71004	Unless WRDE_SHOWERR is set in <i>flags</i> , <i>wordexp()</i> shall redirect <i>stderr</i> to <i>/dev/null</i> for any	
71005	utilities executed as a result of command substitution while expanding <i>words</i> . If	
71006	WRDE_SHOWERR is set, <i>wordexp()</i> may write messages to <i>stderr</i> if syntax errors are detected	
71007	while expanding <i>words</i> .	
71008	The application shall ensure that if WRDE_DOOFFS is set, then <i>pwordexp-&gt;we_offs</i> has the same	
71009	value for each <i>wordexp()</i> call and <i>wordfree()</i> call using a given <i>pwordexp</i> .	

71010 The following constants are defined as error return values:

71011	WRDE_BADCHAR	One of the unquoted characters—<newline>, '   ', ' & ', ' ; ', ' < ', ' > ', ' ( ', ' ) ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
71012		
71013	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
71014	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
71015	WRDE_NOSPACE	Attempt to allocate memory failed.
71016	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.
71017		

#### 71018 RETURN VALUE

71019 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described  
 71020 in <*wordexp.h*>, shall be returned to indicate an error. If *wordexp()* returns the value  
 71021 WRDE\_NOSPACE, then *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall be updated to  
 71022 reflect any words that were successfully expanded. In other cases, they shall not be modified.

71023 The *wordfree()* function shall not return a value.

#### 71024 ERRORS

71025 No errors are defined.

#### 71026 EXAMPLES

71027 None.

#### 71028 APPLICATION USAGE

71029 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's  
 71030 expansions on a word or words obtained from a user. For example, if the application prompts  
 71031 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could  
 71032 respond with anything that would be valid as input to the shell.

71033 The WRDE\_NOCMD flag is provided for applications that, for security or other reasons, want to  
 71034 prevent a user from executing shell commands. Disallowing unquoted shell special characters  
 71035 also prevents unwanted side-effects, such as executing a command or writing a file.

71036 POSIX.1-2008 does not require the *wordexp()* function to be thread-safe if passed an expression  
 71037 referencing an environment variable while any other thread is concurrently modifying any  
 71038 environment variable; see *exec* (on page 772).

#### 71039 RATIONALE

71040 This function was included as an alternative to *glob()*. There had been continuing controversy  
 71041 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*  
 71042 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*  
 71043 (which is faster, but which only performs pathname expansion, without tilde or parameter  
 71044 expansion) this will satisfy the majority of applications.

71045 While *wordexp()* could be implemented entirely as a library routine, it is expected that most  
 71046 implementations run a shell in a subprocess to do the expansion.

71047 Two different approaches have been proposed for how the required information might be  
 71048 presented to the shell and the results returned. They are presented here as examples.

71049 One proposal is to extend the *echo* utility by adding a *-q* option. This option would cause *echo* to  
 71050 add a <backslash> before each <backslash> and <blank> that occurs within an argument. The  
 71051 *wordexp()* function could then invoke the shell as follows:

```
71052 (void) strcpy(buffer, "echo -q");
```

**wordexp()**

```

71053     (void) strcat(buffer, words);
71054     if ((flags & WRDE_SHOWERR) == 0)
71055         (void) strcat(buffer, "2>/dev/null");
71056     f = popen(buffer, "r");

```

71057 The *wordexp()* function would read the resulting output, remove unquoted <backslash>  
 71058 characters, and break into words at unquoted <blank> characters. If the WRDE\_NOCMD flag  
 71059 was set, *wordexp()* would have to scan *words* before starting the subshell to make sure that there  
 71060 would be no command substitution. In any case, it would have to scan *words* for unquoted  
 71061 special characters.

71062 Another proposal is to add the following options to *sh*:

71063 **-w** *wordlist*

71064 This option provides a wordlist expansion service to applications. The words in *wordlist*  
 71065 shall be expanded and the following written to standard output:

- 71066 1. The count of the number of words after expansion, in decimal, followed by a null  
 71067 byte
- 71068 2. The number of bytes needed to represent the expanded words (not including null  
 71069 separators), in decimal, followed by a null byte
- 71070 3. The expanded words, each terminated by a null byte

71071 If an error is encountered during word expansion, *sh* exits with a non-zero status after  
 71072 writing the former to report any words successfully expanded

71073 **-P** Run in "protected" mode. If specified with the **-w** option, no command substitution shall  
 71074 be performed.

71075 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess  
 71076 using *fork()* and executing *sh* using the line:

```

71077 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);

```

71078 after directing standard error to **/dev/null**.

71079 It seemed objectionable for a library routine to write messages to standard error, unless explicitly  
 71080 requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure that no  
 71081 messages are generated, even for commands executed for command substitution. The  
 71082 WRDE\_SHOWERR flag can be specified to request that error messages be written.

71083 The WRDE\_REUSE flag allows the implementation to avoid the expense of freeing and  
 71084 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when  
 71085 WRDE\_REUSE is set.

#### 71086 FUTURE DIRECTIONS

71087 None.

#### 71088 SEE ALSO

71089 [fnmatch\(\)](#), [glob\(\)](#)

71090 XBD [<wordexp.h>](#)

71091 XCU [Chapter 2](#) (on page 2297)

71092 **CHANGE HISTORY**

71093 First released in Issue 4. Derived from the ISO POSIX-2 standard.

71094 **Issue 5**

71095 Moved from POSIX2 C-language Binding to BASE.

71096 **Issue 6**

71097 The normative text is updated to avoid use of the term “must” for application requirements.

71098 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the  
71099 ISO/IEC 9899: 1999 standard.

71100 **Issue 7**

71101 Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**wprintf()**71102 **NAME**71103            **wprintf** — print formatted wide-character output71104 **SYNOPSIS**

71105            #include &lt;stdio.h&gt;

71106            #include &lt;wchar.h&gt;

71107            int wprintf(const wchar\_t \*restrict *format*, ...);71108 **DESCRIPTION**71109            Refer to [fwprintf\(\)](#).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

71110 **NAME**

71111 pwrite, write — write on a file

71112 **SYNOPSIS**

```
71113 #include <unistd.h>
71114 ssize_t pwrite(int fd, const void *buf, size_t nbyte,
71115               off_t offset);
71116 ssize_t write(int fd, const void *buf, size_t nbyte);
```

71117 **DESCRIPTION**

71118 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the  
 71119 file associated with the open file descriptor, *fd*.

71120 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the  
 71121 *write()* function may detect and return errors as described below. In the absence of errors, or if  
 71122 error detection is not performed, the *write()* function shall return zero and have no other results.  
 71123 If *nbyte* is zero and the file is not a regular file, the results are unspecified.

71124 On a regular file or other file capable of seeking, the actual writing of data shall proceed from  
 71125 the position in the file indicated by the file offset associated with *fd*. Before successful return  
 71126 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a  
 71127 regular file, if the position of the last byte written is greater than or equal to the length of the file,  
 71128 the length of the file shall be set to this position plus one.

71129 On a file not capable of seeking, writing shall always take place starting at the current position.  
 71130 The value of a file offset associated with such a device is undefined.

71131 If the O\_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file  
 71132 prior to each write and no intervening file modification operation shall occur between changing  
 71133 the file offset and the write operation.

71134 XSI If a *write()* requests that more bytes be written than there is room for (for example, the file size  
 71135 limit of the process or the physical end of a medium), only as many bytes as there is room for  
 71136 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a  
 71137 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would  
 71138 give a failure return (except as noted below).

71139 XSI If the request would cause the file size to exceed the soft file size limit for the process and there  
 71140 is no room for any bytes to be written, the request shall fail and the implementation shall  
 71141 generate the SIGXFSZ signal for the thread.

71142 If *write()* is interrupted by a signal before it writes any data, it shall return  $-1$  with *errno* set to  
 71143 [EINTR].

71144 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the  
 71145 number of bytes written.

71146 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

71147 After a *write()* to a regular file has successfully returned:

- 71148 • Any successful *read()* from each byte position in the file that was modified by that write  
 71149 shall return the data specified by the *write()* for that position until such byte positions are  
 71150 again modified.
- 71151 • Any subsequent successful *write()* to the same byte position in the file shall overwrite that  
 71152 file data.

## write()

71153		Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the
71154		following exceptions:
71155		• There is no file offset associated with a pipe, hence each write request shall append to the
71156		end of the pipe.
71157		• Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other
71158		processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may
71159		have data interleaved, on arbitrary boundaries, with writes by other processes, whether or
71160		not the O_NONBLOCK flag of the file status flags is set.
71161		• If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on
71162		normal completion it shall return <i>nbyte</i> .
71163		• If the O_NONBLOCK flag is set, <i>write()</i> requests shall be handled differently, in the
71164		following ways:
71165		— The <i>write()</i> function shall not block the thread.
71166		— A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there
71167		is sufficient space available in the pipe, <i>write()</i> shall transfer all the data and return
71168		the number of bytes requested. Otherwise, <i>write()</i> shall transfer no data and return
71169		−1 with <i>errno</i> set to [EAGAIN].
71170		— A write request for more than {PIPE_BUF} bytes shall cause one of the following:
71171		— When at least one byte can be written, transfer what it can and return the
71172		number of bytes written. When all data previously written to the pipe is read, it
71173		shall transfer at least {PIPE_BUF} bytes.
71174		— When no data can be written, transfer no data, and return −1 with <i>errno</i> set to
71175		[EAGAIN].
71176		When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-
71177		blocking writes and cannot accept the data immediately:
71178		• If the O_NONBLOCK flag is clear, <i>write()</i> shall block the calling thread until the data can
71179		be accepted.
71180		• If the O_NONBLOCK flag is set, <i>write()</i> shall not block the thread. If some data can be
71181		written without blocking the thread, <i>write()</i> shall write what it can and return the number
71182		of bytes written. Otherwise, it shall return −1 and set <i>errno</i> to [EAGAIN].
71183		Upon successful completion, where <i>nbyte</i> is greater than 0, <i>write()</i> shall mark for update the last
71184		data modification and last file status change timestamps of the file, and if the file is a regular file,
71185		the S_ISUID and S_ISGID bits of the file mode may be cleared.
71186		For regular files, no data transfer shall occur past the offset maximum established in the open
71187		file description associated with <i>filides</i> .
71188		If <i>filides</i> refers to a socket, <i>write()</i> shall be equivalent to <i>send()</i> with no flags set.
71189	SIO	If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as
71190		defined by synchronized I/O data integrity completion.
71191		If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as
71192		defined by synchronized I/O file integrity completion.
71193	SHM	If <i>filides</i> refers to a shared memory object, the result of the <i>write()</i> function is unspecified.

71194	TYM	If <i>filides</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.
71195	OB XSR	If <i>filides</i> refers to a STREAM, the operation of <i>write()</i> shall be determined by the values of the minimum and maximum <i>nbyte</i> range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If <i>nbyte</i> falls within the packet size range, <i>nbyte</i> bytes shall be written. If <i>nbyte</i> does not fall within the range and the minimum packet size value is 0, <i>write()</i> shall break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If <i>nbyte</i> does not fall within the range and the minimum value is non-zero, <i>write()</i> shall fail with <i>errno</i> set to [ERANGE]. Writing a zero-length buffer ( <i>nbyte</i> is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue <code>L_SWROPT ioctl()</code> to enable zero-length messages to be sent across the pipe or FIFO.
71196		
71197		
71198		
71199		
71200		
71201		
71202		
71203		
71204		
71205		
71206		When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:
71207		
71208		• If <code>O_NONBLOCK</code> is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), <i>write()</i> shall block until data can be accepted.
71209		
71210		
71211		• If <code>O_NONBLOCK</code> is set and the STREAM cannot accept data, <i>write()</i> shall return <code>-1</code> and set <i>errno</i> to [EAGAIN].
71212		
71213		• If <code>O_NONBLOCK</code> is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, <i>write()</i> shall terminate and return the number of bytes written.
71214		
71215		
71216		In addition, <i>write()</i> shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of <i>errno</i> does not reflect the result of <i>write()</i> , but reflects the prior error.
71217		
71218		
71219		The <i>pwrite()</i> function shall be equivalent to <i>write()</i> , except that it writes into a given position and does not change the file offset (regardless of whether <code>O_APPEND</code> is set). The first three arguments to <i>pwrite()</i> are the same as <i>write()</i> with the addition of a fourth argument <i>offset</i> for the desired position inside the file.
71220		
71221		
71222		
71223		<b>RETURN VALUE</b>
71224		Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>filides</i> . This number shall never be greater than <i>nbyte</i> . Otherwise, <code>-1</code> shall be returned and <i>errno</i> set to indicate the error.
71225		
71226		
71227		<b>ERRORS</b>
71228		These functions shall fail if:
71229		[EAGAIN] The <code>O_NONBLOCK</code> flag is set for the file descriptor and the thread would be delayed in the <i>write()</i> operation.
71230		
71231		[EBADF] The <i>filides</i> argument is not a valid file descriptor open for writing.
71232		[EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written.
71233	XSI	
71234		
71235		[EFBIG] The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>filides</i> .
71236		
71237		

## write()

71238	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
71239		
71240	[EIO]	The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
71241		
71242		
71243		
71244	[ENOSPC]	There was no free space remaining on the device containing the file.
71245	[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.
71246		
71247		
71248	OB XSR [ERANGE]	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .
71249		
71250		The <i>write()</i> function shall fail if:
71251	[EAGAIN] or [EWOULDBLOCK]	
71252		The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.
71253		
71254	[ECONNRESET]	A write was attempted on a socket that is not connected.
71255	[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.
71256		
71257		
71258		These functions may fail if:
71259	OB XSR [EINVAL]	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
71260		
71261	[EIO]	A physical I/O error has occurred.
71262	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
71263	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
71264		
71265	OB XSR [ENXIO]	A hangup occurred on the STREAM being written to.
71266	OB XSR	A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, <i>errno</i> is set to the value included in the error message.
71267		
71268		The <i>write()</i> function may fail if:
71269	[EACCES]	A write was attempted on a socket and the calling process does not have appropriate privileges.
71270		
71271	[ENETDOWN]	A write was attempted on a socket and the local network interface used to reach the destination is down.
71272		
71273	[ENETUNREACH]	
71274		A write was attempted on a socket and no route to the network is present.
71275		The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:
71276	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.

71277 [ESPIPE] *fildev* is associated with a pipe or FIFO.

## 71278 EXAMPLES

### 71279 Writing from a Buffer

71280 The following example writes data from the buffer pointed to by *buf* to the file associated with  
71281 the file descriptor *fd*.

```
71282 #include <sys/types.h>
71283 #include <string.h>
71284 ...
71285 char buf[20];
71286 size_t nbytes;
71287 ssize_t bytes_written;
71288 int fd;
71289 ...
71290 strcpy(buf, "This is a test\n");
71291 nbytes = strlen(buf);
71292 bytes_written = write(fd, buf, nbytes);
71293 ...
```

### 71294 APPLICATION USAGE

71295 None.

### 71296 RATIONALE

71297 See also the RATIONALE section in *read()*.

71298 An attempt to write to a pipe or FIFO has several major characteristics:

- 71299 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not  
71300 interleaved with data from any other process. This is useful when there are multiple  
71301 writers sending data to a single reader. Applications need to know how large a write  
71302 request can be expected to be performed atomically. This maximum is called {PIPE\_BUF}.  
71303 This volume of POSIX.1-2008 does not say whether write requests for more than  
71304 {PIPE\_BUF} bytes are atomic, but requires that writes of {PIPE\_BUF} or fewer bytes shall  
71305 be atomic.
- 71306 • *Blocking/immediate*: Blocking is only possible with O\_NONBLOCK clear. If there is enough  
71307 space for all the data requested to be written immediately, the implementation should do  
71308 so. Otherwise, the calling thread may block; that is, pause until enough space is available  
71309 for writing. The effective size of a pipe or FIFO (the maximum amount that can be written  
71310 in one operation without blocking) may vary dynamically, depending on the  
71311 implementation, so it is not possible to specify a fixed value for it.

- 71312 • *Complete/partial/deferred*: A write request:

```
71313 int fildes;
71314 size_t nbyte;
71315 ssize_t ret;
71316 char *buf;
71317 ret = write(fildes, buf, nbyte);
```

71318 may return:

write()

71319 Complete *ret=nbyte*

71320 Partial *ret<nbyte*

71321 This shall never happen if *nbyte*≤{PIPE\_BUF}. If it does happen (with  
 71322 *nbyte*>{PIPE\_BUF}), this volume of POSIX.1-2008 does not guarantee  
 71323 atomicity, even if *ret*≤{PIPE\_BUF}, because atomicity is guaranteed according  
 71324 to the amount *requested*, not the amount *written*.

71325 Deferred: *ret=-1, errno=[EAGAIN]*

71326 This error indicates that a later request may succeed. It does not indicate that  
 71327 it *shall* succeed, even if *nbyte*≤{PIPE\_BUF}, because if no process reads from  
 71328 the pipe or FIFO, the write never succeeds. An application could usefully  
 71329 count the number of times [EAGAIN] is caused by a particular value of  
 71330 *nbyte*>{PIPE\_BUF} and perhaps do later writes with a smaller value, on the  
 71331 assumption that the effective size of the pipe may have decreased.

71332 Partial and deferred writes are only possible with O\_NONBLOCK set.

71333 The relations of these properties are shown in the following tables:

71334 **Write to a Pipe or FIFO with O\_NONBLOCK clear**

71335 <b>Immediately Writable:</b>	<b>None</b>	<b>Some</b>	<i>nbyte</i>
71336 <i>nbyte</i> ≤{PIPE_BUF}	Atomic blocking <i>nbyte</i>	Atomic blocking <i>nbyte</i>	Atomic immediate <i>nbyte</i>
71337 <i>nbyte</i> >{PIPE_BUF}	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>

71339 If the O\_NONBLOCK flag is clear, a write request shall block if the amount writable  
 71340 immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never  
 71341 block.

71342 **Write to a Pipe or FIFO with O\_NONBLOCK set**

71343 <b>Immediately Writable:</b>	<b>None</b>	<b>Some</b>	<i>nbyte</i>
71344 <i>nbyte</i> ≤{PIPE_BUF}	-1, [EAGAIN]	-1, [EAGAIN]	Atomic <i>nbyte</i>
71345 <i>nbyte</i> >{PIPE_BUF}	-1, [EAGAIN]	< <i>nbyte</i> or -1, [EAGAIN]	≤ <i>nbyte</i> or -1, [EAGAIN]

71347 There is no exception regarding partial writes when O\_NONBLOCK is set. With the exception  
 71348 of writing to an empty pipe, this volume of POSIX.1-2008 does not specify exactly when a partial  
 71349 write is performed since that would require specifying internal details of the implementation.  
 71350 Every application should be prepared to handle partial writes when O\_NONBLOCK is set and  
 71351 the requested amount is greater than {PIPE\_BUF}, just as every application should be prepared  
 71352 to handle partial writes on other kinds of file descriptors.

71353 The intent of forcing writing at least one byte if any can be written is to assure that each write  
 71354 makes progress if there is any room in the pipe. If the pipe is empty, {PIPE\_BUF} bytes must be  
 71355 written; if not, at least some progress must have been made.

71356 Where this volume of POSIX.1-2008 requires -1 to be returned and *errno* set to [EAGAIN], most  
 71357 historical implementations return zero (with the O\_NDELAY flag set, which is the historical  
 71358 predecessor of O\_NONBLOCK, but is not itself in this volume of POSIX.1-2008). The error  
 71359 indications in this volume of POSIX.1-2008 were chosen so that an application can distinguish  
 71360 these cases from end-of-file. While *write()* cannot receive an indication of end-of-file, *read()* can,  
 71361 and the two functions have similar return values. Also, some existing systems (for example,  
 71362 Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file

71363 indication; for those systems, a return value of zero from *write()* indicates a successful write of  
71364 an end-of-file indication.

71365 Implementations are allowed, but not required, to perform error checking for *write()* requests of  
71366 zero bytes.

71367 The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can be  
71368 written to a pipe in a single operation) was considered, but rejected, because this concept would  
71369 unnecessarily limit application writing.

71370 See also the discussion of O\_NONBLOCK in *read()*.

71371 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be  
71372 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the  
71373 calls are made by different processes. A similar requirement applies to multiple write operations  
71374 to the same file position. This is needed to guarantee the propagation of data from *write()* calls  
71375 to subsequent *read()* calls. This requirement is particularly significant for networked file  
71376 systems, where some caching schemes violate these semantics.

71377 Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writev()*  
71378 also obey these semantics. A new “high-performance” write analog that did not follow these  
71379 serialization requirements would also be permitted by this wording. This volume of  
71380 POSIX.1-2008 is also silent about any effects of application-level caching (such as that done by  
71381 *stdio*).

71382 This volume of POSIX.1-2008 does not specify the value of the file offset after an error is  
71383 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
71384 meaningless since no file is involved. For errors that are detected immediately, such as  
71385 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
71386 an updated value would be very useful and is the behavior of many implementations.

71387 This volume of POSIX.1-2008 does not specify behavior of concurrent writes to a file from  
71388 multiple processes. Applications should use some form of concurrency control.

#### 71389 FUTURE DIRECTIONS

71390 None.

#### 71391 SEE ALSO

71392 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *read()*, *ulimit()*, *writev()*

71393 XBD <limits.h>, <stropts.h>, <sys/uid.h>, <unistd.h>

#### 71394 CHANGE HISTORY

71395 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 71396 Issue 5

71397 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
71398 Threads Extension.

71399 Large File Summit extensions are added.

71400 The *pwrite()* function is added.

#### 71401 Issue 6

71402 The DESCRIPTION states that the *write()* function does not block the thread. Previously this  
71403 said “process” rather than “thread”.

71404 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
71405 marked as part of the XSI STREAMS Option Group.

**write()**

- 71406 The following new requirements on POSIX implementations derive from alignment with the  
71407 Single UNIX Specification:
- 71408 • The DESCRIPTION now states that if *write()* is interrupted by a signal after it has  
71409 successfully written some data, it returns the number of bytes written. In the POSIX.1-1988  
71410 standard, it was optional whether *write()* returned the number of bytes written, or whether  
71411 it returned  $-1$  with *errno* set to [EINTR]. This is a FIPS requirement.
  - 71412 • The following changes are made to support large files:
    - 71413 — For regular files, no data transfer occurs past the offset maximum established in the  
71414 open file description associated with the *files*.
    - 71415 — A second [EFBIG] error condition is added.
  - 71416 • The [EIO] error condition is added.
  - 71417 • The [EPIPE] error condition is added for when a pipe has only one end open.
  - 71418 • The [ENXIO] optional error condition is added.
- 71419 Text referring to sockets is added to the DESCRIPTION.
- 71420 The following changes were made to align with the IEEE P1003.1a draft standard:
- 71421 • The effect of reading zero bytes is clarified.
- 71422 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
71423 *write()* results are unspecified for typed memory objects.
- 71424 The following error conditions are added for operations on sockets: [EAGAIN],  
71425 [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].
- 71426 The [EIO] error is made optional.
- 71427 The [ENOBUFS] error is added for sockets.
- 71428 The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],  
71429 and [ENETUNREACH].
- 71430 The *writenv()* function is split out into a separate reference page.
- 71431 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the  
71432 ERRORS section from “a SIGPIPE signal is generated to the calling process” to “a SIGPIPE  
71433 signal shall also be sent to the thread”.
- 71434 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the  
71435 RATIONALE.
- 71436 **Issue 7**
- 71437 The *pwrite()* function is moved from the XSI option to the Base.
- 71438 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 71439 SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the  
71440 *pwrite()* function, and to change the use of the phrase “file pointer” to “file offset”.

71441 **NAME**

71442 writev — write a vector

71443 **SYNOPSIS**

```
71444 XSI #include <sys/uio.h>
71445      ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);
```

71446 **DESCRIPTION**

71447 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*  
 71448 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*  
 71449 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than  
 71450 or equal to {IOV\_MAX}, as defined in <limits.h>.

71451 Each *iovec* entry specifies the base address and length of an area in memory from which data  
 71452 should be written. The *writev()* function shall always write a complete area before proceeding to  
 71453 the next.

71454 If *fildev* refers to a regular file and all of the *iov\_len* members in the array pointed to by *iov* are 0,  
 71455 *writev()* shall return 0 and have no other effect. For other file types the behavior is unspecified.

71456 If the sum of the *iov\_len* values is greater than {SSIZE\_MAX}, the operation shall fail and no data  
 71457 shall be transferred.

71458 **RETURN VALUE**

71459 Upon successful completion, *writev()* shall return the number of bytes actually written.  
 71460 Otherwise, it shall return a value of -1, the file pointer shall remain unchanged, and *errno* shall  
 71461 be set to indicate an error.

71462 **ERRORS**71463 Refer to *write()*.71464 In addition, the *writev()* function shall fail if:71465 [EINVAL] The sum of the *iov\_len* values in the *iov* array would overflow an **ssize\_t**.71466 The *writev()* function may fail and set *errno* to:71467 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.71468 **EXAMPLES**71469 **Writing Data from an Array**

71470 The following example writes data from the buffers specified by members of the *iov* array to the  
 71471 file associated with the file descriptor *fd*.

```
71472 #include <sys/types.h>
71473 #include <sys/uio.h>
71474 #include <unistd.h>
71475 ...
71476 ssize_t bytes_written;
71477 int fd;
71478 char *buf0 = "short string\n";
71479 char *buf1 = "This is a longer string\n";
71480 char *buf2 = "This is the longest string in this example\n";
71481 int iovcnt;
71482 struct iovec iov[3];
```

**writev()**

```
71483     iov[0].iov_base = buf0;
71484     iov[0].iov_len = strlen(buf0);
71485     iov[1].iov_base = buf1;
71486     iov[1].iov_len = strlen(buf1);
71487     iov[2].iov_base = buf2;
71488     iov[2].iov_len = strlen(buf2);
71489     ...
71490     iovcnt = sizeof(iov) / sizeof(struct iovec);

71491     bytes_written = writev(fd, iov, iovcnt);
71492     ...
```

**71493 APPLICATION USAGE**

71494 None.

**71495 RATIONALE**

71496 Refer to *write()*.

**71497 FUTURE DIRECTIONS**

71498 None.

**71499 SEE ALSO**

71500 *readv()*, *write()*

71501 XBD [<limits.h>](#), [<sys/uio.h>](#)

**71502 CHANGE HISTORY**

71503 First released in Issue 4, Version 2.

**71504 Issue 6**

71505 Split out from the *write()* reference page.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wscanf()**71506 **NAME**

71507           wscanf — convert formatted wide-character input

71508 **SYNOPSIS**

71509           #include &lt;stdio.h&gt;

71510           #include &lt;wchar.h&gt;

71511           int wscanf(const wchar\_t \*restrict format, ...);

71512 **DESCRIPTION**71513           Refer to *fwscanf()*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**y0()**71514 **NAME**71515 `y0, y1, yn` — Bessel functions of the second kind71516 **SYNOPSIS**

```
71517 xSI #include <math.h>
71518 double y0(double x);
71519 double y1(double x);
71520 double yn(int n, double x);
```

71521 **DESCRIPTION**

71522 The `y0()`, `y1()`, and `yn()` functions shall compute Bessel functions of  $x$  of the second kind of  
 71523 orders 0, 1, and  $n$ , respectively.

71524 An application wishing to check for error situations should set `errno` to zero and call  
 71525 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 71526 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 71527 zero, an error has occurred.

71528 **RETURN VALUE**

71529 Upon successful completion, these functions shall return the relevant Bessel value of  $x$  of the  
 71530 second kind.

71531 If  $x$  is NaN, NaN shall be returned.

71532 If the  $x$  argument to these functions is negative, `-HUGE_VAL` or NaN shall be returned, and a  
 71533 domain error may occur.

71534 If  $x$  is 0.0, `-HUGE_VAL` shall be returned and a pole error may occur.

71535 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

71536 If the correct result would cause overflow, `-HUGE_VAL` or 0.0 shall be returned and a range  
 71537 error may occur.

71538 **ERRORS**

71539 These functions may fail if:

71540 **Domain Error** The value of  $x$  is negative.

71541 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
 71542 then `errno` shall be set to [EDOM]. If the integer expression (`math_errhandling`  
 71543 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception  
 71544 shall be raised.

71545 **Pole Error** The value of  $x$  is zero.

71546 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
 71547 then `errno` shall be set to [ERANGE]. If the integer expression  
 71548 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the divide-by-zero  
 71549 floating-point exception shall be raised.

71550 **Range Error** The correct result would cause overflow.

71551 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
 71552 then `errno` shall be set to [ERANGE]. If the integer expression  
 71553 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the overflow  
 71554 floating-point exception shall be raised.

71555	Range Error	The value of $x$ is too large in magnitude, or the correct result would cause underflow.
71556		
71557		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [ERANGE]. If the integer expression ( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.
71558		
71559		
71560		
71561	<b>EXAMPLES</b>	
71562	None.	
71563	<b>APPLICATION USAGE</b>	
71564	On error, the expressions ( <i>math_errhandling</i> & MATH_ERRNO) and ( <i>math_errhandling</i> & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.	
71565		
71566	<b>RATIONALE</b>	
71567	None.	
71568	<b>FUTURE DIRECTIONS</b>	
71569	None.	
71570	<b>SEE ALSO</b>	
71571	<a href="#"><i>feclearexcept()</i></a> , <a href="#"><i>fetestexcept()</i></a> , <a href="#"><i>isnan()</i></a> , <a href="#"><i>j0()</i></a>	
71572	XBD Section 4.19 (on page 116), <a href="#"><b>&lt;math.h&gt;</b></a>	
71573	<b>CHANGE HISTORY</b>	
71574	First released in Issue 1. Derived from Issue 1 of the SVID.	
71575	<b>Issue 5</b>	
71576	The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.	
71577		
71578	<b>Issue 6</b>	
71579	The normative text is updated to avoid use of the term “must” for application requirements.	
71580	The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899:1999 standard.	
71581		
71582	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN VALUE and ERRORS sections. The changes are made for consistency with the general rules stated in “Treatment of Error Conditions for Mathematical Functions” in the Base Definitions volume of POSIX.1-2008.	
71583		
71584		
71585		

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

71586

 *Technical Standard*

71587

**Vol. 3:**

71588

**Shell and Utilities, Issue 7**

71589

*The Open Group*

71590

*The Institute of Electrical and Electronics Engineers, Inc.*

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

71591

Chapter 1

71592

*Introduction*

71593

The Shell and Utilities volume of POSIX.1-2008 describes the commands and utilities offered to application programs by POSIX-conformant systems.

71594

71595 **1.1 Relationship to Other Documents**71596 **1.1.1 System Interfaces**

71597 This subsection describes some of the features provided by the System Interfaces volume of  
71598 POSIX.1-2008 that are assumed to be globally available on all systems conforming to this volume  
71599 of POSIX.1-2008. This subsection does not attempt to detail all of the features defined in the  
71600 System Interfaces volume of POSIX.1-2008 that are required by all of the utilities defined in this  
71601 volume of POSIX.1-2008; the utility and function descriptions point out additional functionality  
71602 required to provide the corresponding specific features needed by each.

71603 The following subsections describe frequently used concepts. Many of these concepts are  
71604 described in the Base Definitions volume of POSIX.1-2008. Utility and function description  
71605 statements override these defaults when appropriate.

71606 **1.1.1.1 Process Attributes**

71607 The following process attributes, as described in the System Interfaces volume of POSIX.1-2008,  
71608 are assumed to be supported for all processes in this volume of POSIX.1-2008:

71609	Controlling Terminal	Real Group ID
71610	Current Working Directory	Real User ID
71611	Effective Group ID	Root Directory
71612	Effective User ID	Saved Set-Group-ID
71613	File Descriptors	Saved Set-User-ID
71614	File Mode Creation Mask	Session Membership
71615	Process Group ID	Supplementary Group IDs
71616	Process ID	

71617 A conforming implementation may include additional process attributes.

71618 **1.1.1.2 Concurrent Execution of Processes**

71619 The following functionality of the *fork()* function defined in the System Interfaces volume of  
71620 POSIX.1-2008 shall be available on all systems conforming to this volume of POSIX.1-2008:

- 71621 1. Independent processes shall be capable of executing independently without either  
71622 process terminating.

- 71623 2. A process shall be able to create a new process with all of the attributes referenced in  
 71624 [Section 1.1.1.1](#) (on page 2279), determined according to the semantics of a call to the *fork()*  
 71625 function defined in the System Interfaces volume of POSIX.1-2008 followed by a call in  
 71626 the child process to one of the *exec* functions defined in the System Interfaces volume of  
 71627 POSIX.1-2008.
- 71628 **1.1.1.3 File Access Permissions**
- 71629 The file access control mechanism described by XBD [Section 4.4](#) (on page 108) shall apply to all  
 71630 files on an implementation conforming to this volume of POSIX.1-2008.
- 71631 **1.1.1.4 File Read, Write, and Creation**
- 71632 If a file that does not exist is to be written, it shall be created as described below, unless the  
 71633 utility description states otherwise.
- 71634 When a file that does not exist is created, the following features defined in the System Interfaces  
 71635 volume of POSIX.1-2008 shall apply unless the utility or function description states otherwise:
- 71636 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 71637 2. The group ID of the file shall be set to the effective group ID of the calling process or the  
 71638 group ID of the directory in which the file is being created.
- 71639 3. If the file is a regular file, the permission bits of the file shall be set to:
- 71640  $S\_IROTH \mid S\_IWOTH \mid S\_IRGRP \mid S\_IWGRP \mid S\_IRUSR \mid S\_IWUSR$
- 71641 (see the description of *File Modes* in XBD [Chapter 13](#) (on page 219), `<sys/stat.h>`) except  
 71642 that the bits specified by the file mode creation mask of the process shall be cleared. If the  
 71643 file is a directory, the permission bits shall be set to:
- 71644  $S\_IRWXU \mid S\_IRWXG \mid S\_IRWXO$
- 71645 except that the bits specified by the file mode creation mask of the process shall be  
 71646 cleared.
- 71647 4. The last data access, last data modification, and last file status change timestamps of the  
 71648 file shall be updated as specified in XBD [Section 4.8](#) (on page 109).
- 71649 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length  
 71650 zero.
- 71651 6. If the file is a symbolic link, the effect shall be undefined unless the `{POSIX2_SYMLINKS}`  
 71652 variable is in effect for the directory in which the symbolic link would be created.
- 71653 7. Unless otherwise specified, the file created shall be a regular file.
- 71654 When an attempt is made to create a file that already exists, the utility shall take the action  
 71655 indicated in [Table 1-1](#) (on page 2281) corresponding to the type of the file the utility is trying to  
 71656 create and the type of the existing file, unless the utility description states otherwise.

71657

**Table 1-1** Actions when Creating a File that Already Exists

Existing Type	New Type											Function Creating New
	B	C	D	F	L	M	P	Q	R	S	T	
A <i>fattach()</i> -ed STREAM	F	F	F	F	F	—	—	—	OF	—	U	N/A
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
D Directory	F	F	F	F	F	—	—	—	F	—	U	<i>mkdir()</i>
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	U	<i>mkfifo()</i>
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	U	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	—	—	—	—	—	U	<i>shm_open()</i>
P Semaphore	F	F	F	F	F	—	—	—	—	—	U	<i>sem_open()</i>
Q Message Queue	F	F	F	F	F	—	—	—	—	—	U	<i>mq_open()</i>
R Regular File	F	F	F	F	F	—	—	—	RF	—	U	<i>open()</i>
S Socket	F	F	F	F	F	—	—	—	—	—	U	<i>bind()</i>
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*

71672

The following codes are used in Table 1-1:

71673

**F** Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

71674

71675

71676

**FL** Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

71677

71678

71679

71680

**O** Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

71681

71682

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.

71683

71684

2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

71685

71686

**OF** The named file shall be opened with the consequences defined for that file type.

71687

**RF** Regular file. When attempting to create a regular file, and the existing file is a regular file:

71688

1. The user ID, group ID, and permission bits of the file shall not be changed.

71689

2. The file shall be truncated to zero length.

71690

3. The last data modification and last file status change timestamps shall be marked for update.

71691

71692

— The effect is implementation-defined unless specified by the utility description.

71693

**U** The effect is unspecified unless specified by the utility description.

71694

\* There is no portable way to create a file of this type.

71695

\*\* Not portable.

71696

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the `open()` function defined in the System Interfaces volume of POSIX.1-2008.

71697

71698

71699 When a file is to be read or written, the file shall be opened with an access mode corresponding  
 71700 to the operation to be performed. If file access permissions deny access, the requested operation  
 71701 shall fail.

71702 1.1.1.5 File Removal

71703 When a directory that is the root directory or current working directory of any process is  
 71704 removed, the effect is implementation-defined. If file access permissions deny access, the  
 71705 requested operation shall fail. Otherwise, when a file is removed:

- 71706 1. Its directory entry shall be removed from the file system.
- 71707 2. The link count of the file shall be decremented.
- 71708 3. If the file is an empty directory (see XBD Section 3.144, on page 56):
  - 71709 a. If no process has the directory open, the space occupied by the directory shall be  
 71710 freed and the directory shall no longer be accessible.
  - 71711 b. If one or more processes have the directory open, the directory contents shall be  
 71712 preserved until all references to the file have been closed.
- 71713 4. If the file is a directory that is not empty, the last file status change timestamp shall be  
 71714 marked for update.
- 71715 5. If the file is not a directory:
  - 71716 a. If the link count becomes zero:
    - 71717 i. If no process has the file open, the space occupied by the file shall be freed  
 71718 and the file shall no longer be accessible.
    - 71719 ii. If one or more processes have the file open, the file contents shall be  
 71720 preserved until all references to the file have been closed.
  - 71721 b. If the link count is not reduced to zero, the last file status change timestamp shall  
 71722 be marked for update.
- 71723 6. The last data modification and last file status change timestamps of the containing  
 71724 directory shall be marked for update.

71725 1.1.1.6 File Time Values

71726 All files shall have the three time values described by XBD Section 4.8 (on page 109).

71727 1.1.1.7 File Contents

71728 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of  
 71729 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the  
 71730 following operations defined in the System Interfaces volume of POSIX.1-2008:

```
71731 while (read (fildes, buf, nbytes) > 0)
71732     ;
```

71733 If the file is indicated by a *pathname*, the file descriptor shall be determined by the  
 71734 equivalent of the following operation defined in the System Interfaces volume of POSIX.1-2008:

```
71735 fildes = open (pathname, O_RDONLY);
```

71736 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data

- 71737 returned by *read()* would vary with different values, the value shall be one that results in the  
71738 most data being returned.
- 71739 If the *read()* function calls would return an error, it is unspecified whether the contents of the file  
71740 are considered to include any data from offsets in the file beyond where the error would be  
71741 returned.
- 71742 **1.1.1.8 Pathname Resolution**
- 71743 The pathname resolution algorithm, described by XBD [Section 4.12](#) (on page 111), shall be used  
71744 by implementations conforming to this volume of POSIX.1-2008; see also XBD [Section 4.5](#) (on  
71745 page 108).
- 71746 **1.1.1.9 Changing the Current Working Directory**
- 71747 When the current working directory (see XBD [Section 3.122](#), on page 53) is to be changed, unless  
71748 the utility or function description states otherwise, the operation shall succeed unless a call to  
71749 the *chdir()* function defined in the System Interfaces volume of POSIX.1-2008 would fail when  
71750 invoked with the new working directory pathname as its argument.
- 71751 **1.1.1.10 Establish the Locale**
- 71752 The functionality of the *setlocale()* function defined in the System Interfaces volume of  
71753 POSIX.1-2008 shall be available on all systems conforming to this volume of POSIX.1-2008; that  
71754 is, utilities that require the capability of establishing an international operating environment  
71755 shall be permitted to set the specified category of the international environment.
- 71756 **1.1.1.11 Actions Equivalent to Functions**
- 71757 Some utility descriptions specify that a utility performs actions equivalent to a function defined  
71758 in the System Interfaces volume of POSIX.1-2008. Such specifications require only that the  
71759 external effects be equivalent, not that any effect within the utility and visible only to the utility  
71760 be equivalent.
- 71761 **1.1.2 Concepts Derived from the ISO C Standard**
- 71762 Some of the standard utilities perform complex data manipulation using their own procedure  
71763 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS  
71764 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type  
71765 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,  
71766 as described in the following sections. Note that there is no requirement that the standard  
71767 utilities be implemented in any particular programming language.
- 71768 **1.1.2.1 Arithmetic Precision and Operations**
- 71769 Integer variables and constants, including the values of operands and option-arguments, used  
71770 by the standard utilities listed in this volume of POSIX.1-2008 shall be implemented as  
71771 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as  
71772 equivalent to the ISO C standard **double** type. Conversions between types shall be as described  
71773 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned  
71774 by the input to the application.

71775 Arithmetic operators and control flow keywords shall be implemented as equivalent to those in  
 71776 the cited ISO C standard section, as listed in Table 1-2.

71777 **Table 1-2** Selected ISO C Standard Operators and Control Flow Keywords

Operation	ISO C Standard Equivalent Reference
( )	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
expr?expr:expr	Section 6.5.15, Conditional Operator
=, *=, /=, %= <<=, >>=, &=, ^=,  =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

71818 The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5,  
71819 Expressions, of the ISO C standard.

71820 1.1.2.2 *Mathematical Functions*

71821 Any mathematical functions with the same names as those in the following sections of the ISO C  
71822 standard:

- 71823 • Section 7.12, Mathematics, <math.h>
- 71824 • Section 7.20.2, Pseudo-Random Sequence Generation Functions

71825 shall be implemented to return the results equivalent to those returned from a call to the  
71826 corresponding function described in the ISO C standard.

71827 **1.2 Utility Limits**

71828 This section lists magnitude limitations imposed by a specific implementation. The braces  
71829 notation, {LIMIT}, is used in this volume of POSIX.1-2008 to indicate these values, but the braces  
71830 are not part of the name.

71831 **Table 1-3** Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see the <b>border_start</b> keyword in XBD Section 7.3.2 (on page 146).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see XBD Section 9.3.6 (on page 186).	255

71857 The values specified in Table 1-3 represent the lowest values conforming implementations shall  
71858 provide and, consequently, the largest values on which an application can rely without further

71859 enquiries, as described below. These values shall be accessible to applications via the *getconf*  
 71860 utility (see *getconf*, on page 2772).

71861 Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3  
 71862 (on page 2285). These possibly more liberal values are accessible using the symbols in Table 1-4.

71863 The *sysconf()* function defined in the System Interfaces volume of POSIX.1-2008 or the *getconf*  
 71864 utility return the value of each symbol on each specific implementation. The value so retrieved is  
 71865 the largest, or most liberal, value that is available throughout the session lifetime, as determined  
 71866 at session creation. The literal names shown in the table apply only to the *getconf* utility; the  
 71867 high-level language binding describes the exact form of each name to be used by the interfaces  
 71868 in that binding.

71869 All numeric limits defined by the System Interfaces volume of POSIX.1-2008, such as  
 71870 {PATH\_MAX}, shall also apply to this volume of POSIX.1-2008. All the utilities defined by this  
 71871 volume of POSIX.1-2008 are implicitly limited by these values, unless otherwise noted in the  
 71872 utility descriptions.

71873 It is not guaranteed that the application can actually reach the specified limit of an  
 71874 implementation in any given case, or at all, as a lack of virtual memory or other resources may  
 71875 prevent this. The limit value indicates only that the implementation does not specifically impose  
 71876 any arbitrary, more restrictive limit.

71877 **Table 1-4** Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>ibase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <b>LC_COLLATE order</b> keyword in the locale definition file; see the <b>order_start</b> keyword in XBD Section 7.3.2 (on page 146).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}

	Name	Description	Minimum Value
71901			
71902	{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
71903			
71904			
71905			
71906			
71907			
71908			
71909			
71910			
71911			
71912	{RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see XBD Section 9.3.6 (on page 186).	{POSIX2_RE_DUP_MAX}
71913			
71914			
71915			
71916			

The following value may be a constant within an implementation or may vary from one pathname to another.

71917 {POSIX2\_SYMLINKS}

71918 When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2\_SYMLINKS} is undefined.

### 71922 1.3 Grammar Conventions

71923 Portions of this volume of POSIX.1-2008 are expressed in terms of a special grammar notation. It  
 71924 is used to portray the complex syntax of certain program input. The grammar is based on the  
 71925 syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input,  
 71926 suitable for program use; the lexical processing and all semantic requirements are described only  
 71927 in textual form. The grammar is not based on source used in any traditional implementation and  
 71928 has not been tested with the semantic code that would normally be required to accompany it.  
 71929 Furthermore, there is no implication that the partial *yacc* code presented represents the most  
 71930 efficient, or only, means of supporting the complex syntax within the utility. Implementations  
 71931 may use other programming languages or algorithms, as long as the syntax supported is the  
 71932 same as that represented by the grammar.

71933 The following typographical conventions are used in the grammar; they have no significance  
 71934 except to aid in reading.

- 71935 • The identifiers for the reserved words of the language are shown with a leading capital  
 71936 letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- 71937 • The identifiers for terminals in the grammar are all named with uppercase letters and  
 71938 underscores; for example, **NEWLINE**, **ASSIGN\_OP**, **NAME**.
- 71939 • The identifiers for non-terminals are all lowercase.

## 71940 1.4 Utility Description Defaults

71941 This section describes all of the subsections used within the utility descriptions, including:

- 71942 • Intended usage of the section
- 71943 • Global defaults that affect all the standard utilities
- 71944 • The meanings of notations used in this volume of POSIX.1-2008 that are specific to
- 71945 individual utility sections

### 71946 NAME

71947 This section gives the name or names of the utility and briefly states its purpose.

### 71948 SYNOPSIS

71949 The SYNOPSIS section summarizes the syntax of the calling sequence for the utility,  
71950 including options, option-arguments, and operands. Standards for utility naming are  
71951 described in XBD Section 12.2 (on page 215); for describing the utility's arguments in  
71952 XBD Section 12.1 (on page 213).

### 71953 DESCRIPTION

71954 The DESCRIPTION section describes the actions of the utility. If the utility has a very  
71955 complex set of subcommands or its own procedural language, an EXTENDED  
71956 DESCRIPTION section is also provided. Most explanations of optional functionality are  
71957 omitted here, as they are usually explained in the OPTIONS section.

71958 As stated in Section 1.1.1.11 (on page 2283), some functions are described in terms of  
71959 equivalent functionality. When specific functions are cited, the implementation shall  
71960 provide equivalent functionality including side-effects associated with successful  
71961 execution of the function. The treatment of errors and intermediate results from the  
71962 individual functions cited is generally not specified by this volume of POSIX.1-2008.  
71963 See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all  
71964 actions associated with errors encountered by the utility.

### 71965 OPTIONS

71966 The OPTIONS section describes the utility options and option-arguments, and how  
71967 they modify the actions of the utility. Standard utilities that have options either fully  
71968 comply with XBD Section 12.2 (on page 215) or describe all deviations. Apparent  
71969 disagreements between functionality descriptions in the OPTIONS and DESCRIPTION  
71970 (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS  
71971 section.

71972 Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility  
71973 Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of  
71974 POSIX.1-2008; implementation extensions should also conform to the guidelines, but  
71975 may allow exceptions for historical practice.

71976 Unless otherwise stated in the utility description, when given an option unrecognized  
71977 by the implementation, or when a required option-argument is not provided, standard  
71978 utilities shall issue a diagnostic message to standard error and exit with a non-zero exit  
71979 status.

71980 All utilities in this volume of POSIX.1-2008 shall be capable of processing arguments  
71981 using eight-bit transparency.

71982 **Default Behavior:** When this section is listed as "None.", it means that the  
71983 implementation need not support any options. Standard utilities that do not accept  
71984 options, but that do accept operands, shall recognize "--" as a first argument to be  
71985 discarded.

71986 The requirement for recognizing "--" is because conforming applications need a way  
 71987 to shield their operands from any arbitrary options that the implementation may  
 71988 provide as an extension. For example, if the standard utility *foo* is listed as taking no  
 71989 options, and the application needed to give it a pathname with a leading <hyphen>, it  
 71990 could safely do it as:

71991 `foo -- -myfile`

71992 and avoid any problems with `-m` used as an extension.

### 71993 OPERANDS

71994 The OPERANDS section describes the utility operands, and how they affect the actions  
 71995 of the utility. Apparent disagreements between functionality descriptions in the  
 71996 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be  
 71997 resolved in favor of the OPERANDS section.

71998 If an operand naming a file can be specified as `'-'`, which means to use the standard  
 71999 input instead of a named file, this is explicitly stated in this section. Unless otherwise  
 72000 stated, the use of multiple instances of `'-'` to mean standard input in a single  
 72001 command produces unspecified results.

72002 Unless otherwise stated, the standard utilities that accept operands shall process those  
 72003 operands in the order specified in the command line.

72004 **Default Behavior:** When this section is listed as "None.", it means that the  
 72005 implementation need not support any operands.

### 72006 STDIN

72007 The STDIN section describes the standard input of the utility. This section is frequently  
 72008 merely a reference to the following section, as many utilities treat standard input and  
 72009 input files in the same manner. Unless otherwise stated, all restrictions described in the  
 72010 INPUT FILES section shall apply to this section as well.

72011 Use of a terminal for standard input can cause any of the standard utilities that read  
 72012 standard input to stop when used in the background. For this reason, applications  
 72013 should not use interactive features in scripts to be placed in the background.

72014 The specified standard input format of the standard utilities shall not depend on the  
 72015 existence or value of the environment variables defined in this volume of POSIX.1-2008,  
 72016 except as provided by this volume of POSIX.1-2008.

72017 **Default Behavior:** When this section is listed as "Not used.", it means that the standard  
 72018 input shall not be read when the utility is used as described by this volume of  
 72019 POSIX.1-2008.

### 72020 INPUT FILES

72021 The INPUT FILES section describes the files, other than the standard input, used as  
 72022 input by the utility. It includes files named as operands and option-arguments as well  
 72023 as other files that are referred to, such as start-up and initialization files, databases, and  
 72024 so on. Commonly-used files are generally described in one place and cross-referenced  
 72025 by other utilities.

72026 All utilities in this volume of POSIX.1-2008 shall be capable of processing input files  
 72027 using eight-bit transparency.

72028 When a standard utility reads a seekable input file and terminates without an error  
 72029 before it reaches end-of-file, the utility shall ensure that the file offset in the open file  
 72030 description is properly positioned just past the last byte processed by the utility. For  
 72031 files that are not seekable, the state of the file offset in the open file description for that

72032 file is unspecified. A conforming application shall not assume that the following three  
72033 commands are equivalent:

```
72034 tail -n +2 file
72035 (sed -n 1q; cat) < file
72036 cat file | (sed -n 1q; cat)
```

72037 The second command is equivalent to the first only when the file is seekable. The third  
72038 command leaves the file offset in the open file description in an unspecified state. Other  
72039 utilities, such as *head*, *read*, and *sh*, have similar properties.

72040 Some of the standard utilities, such as filters, process input files a line or a block at a  
72041 time and have no restrictions on the maximum input file size. Some utilities may have  
72042 size limitations that are not as obvious as file space or memory limitations. Such  
72043 limitations should reflect resource limitations of some sort, not arbitrary limits set by  
72044 implementors. Implementations shall document those utilities that are limited by  
72045 constraints other than file system space, available memory, and other limits specifically  
72046 cited by this volume of POSIX.1-2008, and identify what the constraint is and indicate a  
72047 way of estimating when the constraint would be reached. Similarly, some utilities  
72048 descend the directory tree (recursively). Implementations shall also document any  
72049 limits that they may have in descending the directory tree that are beyond limits cited  
72050 by this volume of POSIX.1-2008.

72051 When an input file is described as a “text file”, the utility produces undefined results if  
72052 given input that is not from a text file, unless otherwise stated. Some utilities (for  
72053 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>  
72054 convention; unless otherwise stated, the utility need not be able to accumulate more  
72055 than {LINE\_MAX} bytes from a set of multiple, continued input lines. Thus, for a  
72056 conforming application the total of all the continued lines in a set cannot exceed  
72057 {LINE\_MAX}. If a utility using the escaped <newline> convention detects an end-of-  
72058 file condition immediately after an escaped <newline>, the results are unspecified.

72059 Record formats are described in a notation similar to that used by the C-language  
72060 function, *printf*(. See XBD Chapter 5 (on page 121) for a description of this notation.  
72061 The format description is intended to be sufficiently rigorous to allow other  
72062 applications to generate these input files. However, since <blank>s can legitimately be  
72063 included in some of the fields described by the standard utilities, particularly in locales  
72064 other than the POSIX locale, this intent is not always realized.

72065 **Default Behavior:** When this section is listed as “None.”, it means that no input files  
72066 are required to be supplied when the utility is used as described by this volume of  
72067 POSIX.1-2008.

## 72068 ENVIRONMENT VARIABLES

72069 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s  
72070 execution.

72071 The entire manner in which environment variables described in this volume of  
72072 POSIX.1-2008 affect the behavior of each utility is described in the ENVIRONMENT  
72073 VARIABLES section for that utility, in conjunction with the global effects of the *LANG*,  
72074 XSI *LC\_ALL*, and *NLSPATH* environment variables described in XBD Chapter 8 (on page  
72075 173). The existence or value of environment variables described in this volume of  
72076 POSIX.1-2008 shall not otherwise affect the specified behavior of the standard utilities.  
72077 Any effects of the existence or value of environment variables not described by this  
72078 volume of POSIX.1-2008 upon the standard utilities are unspecified.

72079 For those standard utilities that use environment variables as a means for selecting a

72080 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to  
72081 the path search described for *PATH* in XBD Chapter 8 (on page 173).

72082 All utilities in this volume of POSIX.1-2008 shall be capable of processing environment  
72083 variable names and values using eight-bit transparency.

72084 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of  
72085 the utility is not directly affected by environment variables described by this volume of  
72086 POSIX.1-2008 when the utility is used as described by this volume of POSIX.1-2008.

#### 72087 ASYNCHRONOUS EVENTS

72088 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as  
72089 signals and what signals are caught.

72090 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard  
72091 action for all other signals; see Section 1.4 (on page 2288)” it means that the action taken  
72092 as a result of the signal shall be one of the following:

- 72093 1. The action shall be that inherited from the parent according to the rules of  
72094 inheritance of signal actions defined in the System Interfaces volume of  
72095 POSIX.1-2008.
- 72096 2. When no action has been taken to change the default, the default action shall be  
72097 that specified by the System Interfaces volume of POSIX.1-2008.
- 72098 3. The result of the utility’s execution is as if default actions had been taken.

72099 A utility is permitted to catch a signal, perform some additional processing (such as  
72100 deleting temporary files), restore the default signal action (or action inherited from the  
72101 parent process), and resignal itself.

#### 72102 STDOUT

72103 The STDOUT section completely describes the standard output of the utility. This  
72104 section is frequently merely a reference to the following section, OUTPUT FILES,  
72105 because many utilities treat standard output and output files in the same manner.

72106 Use of a terminal for standard output may cause any of the standard utilities that write  
72107 standard output to stop when used in the background. For this reason, applications  
72108 should not use interactive features in scripts to be placed in the background.

72109 Record formats are described in a notation similar to that used by the C-language  
72110 function, *printf()*. See XBD Chapter 5 (on page 121) for a description of this notation.

72111 The specified standard output of the standard utilities shall not depend on the  
72112 existence or value of the environment variables defined in this volume of POSIX.1-2008,  
72113 except as provided by this volume of POSIX.1-2008.

72114 Some of the standard utilities describe their output using the verb *display*, defined in  
72115 XBD Section 3.133 (on page 54). Output described in the STDOUT sections of such  
72116 utilities may be produced using means other than standard output. When standard  
72117 output is directed to a terminal, the output described shall be written directly to the  
72118 terminal. Otherwise, the results are undefined.

72119 **Default Behavior:** When this section is listed as “Not used.”, it means that the standard  
72120 output shall not be written when the utility is used as described by this volume of  
72121 POSIX.1-2008.

#### 72122 STDERR

72123 The STDERR section describes the standard error output of the utility. Only those  
72124 messages that are purposely sent by the utility are described.

72125 Use of a terminal for standard error may cause any of the standard utilities that write  
 72126 standard error output to stop when used in the background. For this reason,  
 72127 applications should not use interactive features in scripts to be placed in the  
 72128 background.

72129 The format of diagnostic messages for most utilities is unspecified, but the language  
 72130 and cultural conventions of diagnostic and informative messages whose format is  
 72131 unspecified by this volume of POSIX.1-2008 should be affected by the setting of  
 72132 XSI `LC_MESSAGES` and `NLSPATH`.

72133 The specified standard error output of standard utilities shall not depend on the  
 72134 existence or value of the environment variables defined in this volume of POSIX.1-2008,  
 72135 except as provided by this volume of POSIX.1-2008.

72136 **Default Behavior:** When this section is listed as “The standard error shall be used only  
 72137 for diagnostic messages.”, it means that, unless otherwise stated, the diagnostic  
 72138 messages shall be sent to the standard error only when the exit status indicates that an  
 72139 error occurred and the utility is used as described by this volume of POSIX.1-2008.

72140 When this section is listed as “Not used.”, it means that the standard error shall not be  
 72141 used when the utility is used as described in this volume of POSIX.1-2008.

## 72142 OUTPUT FILES

72143 The OUTPUT FILES section completely describes the files created or modified by the  
 72144 utility. Temporary or system files that are created for internal usage by this utility or  
 72145 other parts of the implementation (for example, spool, log, and audit files) are not  
 72146 described in this, or any, section. The utilities creating such files and the names of such  
 72147 files are unspecified. If applications are written to use temporary or intermediate files,  
 72148 they should use the `TMPDIR` environment variable, if it is set and represents an  
 72149 accessible directory, to select the location of temporary files.

72150 Implementations shall ensure that temporary files, when used by the standard utilities,  
 72151 are named so that different utilities or multiple instances of the same utility can operate  
 72152 simultaneously without regard to their working directories, or any other process  
 72153 characteristic other than process ID. There are two exceptions to this rule:

- 72154 1. Resources for temporary files other than the name space (for example, disk  
 72155 space, available directory entries, or number of processes allowed) are not  
 72156 guaranteed.
- 72157 2. Certain standard utilities generate output files that are intended as input for  
 72158 other utilities (for example, `lex` generates `lex.yy.c`), and these cannot have unique  
 72159 names. These cases are explicitly identified in the descriptions of the respective  
 72160 utilities.

72161 Any temporary file created by the implementation shall be removed by the  
 72162 implementation upon a utility’s successful exit, exit because of errors, or before  
 72163 termination by any of the `SIGHUP`, `SIGINT`, or `SIGTERM` signals, unless specified  
 72164 otherwise by the utility description.

72165 Receipt of the `SIGQUIT` signal should generally cause termination (unless in some  
 72166 debugging mode) that would bypass any attempted recovery actions.

72167 Record formats are described in a notation similar to that used by the C-language  
 72168 function, `printf()`; see XBD Chapter 5 (on page 121) for a description of this notation.

72169 **Default Behavior:** When this section is listed as “None.”, it means that no files are  
 72170 created or modified as a consequence of direct action on the part of the utility when the  
 72171 utility is used as described by this volume of POSIX.1-2008. However, the utility may

72172 create or modify system files, such as log files, that are outside the utility's normal  
72173 execution environment.

#### 72174 EXTENDED DESCRIPTION

72175 The EXTENDED DESCRIPTION section provides a place for describing the actions of  
72176 very complicated utilities, such as text editors or language processors, which typically  
72177 have elaborate command languages.

72178 **Default Behavior:** When this section is listed as "None.", no further description is  
72179 necessary.

#### 72180 EXIT STATUS

72181 The EXIT STATUS section describes the values the utility shall return to the calling  
72182 program, or shell, and the conditions that cause these values to be returned. Usually,  
72183 utilities return zero for successful completion and values greater than zero for various  
72184 error conditions. If specific numeric values are listed in this section, the system shall  
72185 use those values for the errors described. In some cases, status values are listed more  
72186 loosely, such as >0. A strictly conforming application shall not rely on any specific  
72187 value in the range shown and shall be prepared to receive any value in the range.

72188 For example, a utility may list zero as a successful return, 1 as a failure for a specific  
72189 reason, and >1 as "an error occurred". In this case, unspecified conditions may cause a  
72190 2 or 3, or other value, to be returned. A conforming application should be written so  
72191 that it tests for successful exit status values (zero in this case), rather than relying upon  
72192 the single specific error value listed in this volume of POSIX.1-2008. In that way, it has  
72193 maximum portability, even on implementations with extensions.

72194 Unspecified error conditions may be represented by specific values not listed in this  
72195 volume of POSIX.1-2008.

#### 72196 CONSEQUENCES OF ERRORS

72197 The CONSEQUENCES OF ERRORS section describes the effects on the environment,  
72198 file systems, process state, and so on, when error conditions occur. It does not describe  
72199 error messages produced or exit status values used.

72200 The many reasons for failure of a utility are generally not specified by the utility  
72201 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of  
72202 options, arguments, or environment variables; invalid usage of the complex syntaxes  
72203 expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating,  
72204 reading, or writing files; or difficulties associated with the privileges of the process.

72205 The following shall apply to each utility, unless otherwise stated:

- 72206 • If the requested action cannot be performed on an operand representing a file,  
72207 directory, user, process, and so on, the utility shall issue a diagnostic message to  
72208 standard error and continue processing the next operand in sequence, but the  
72209 final exit status shall be returned as non-zero.

72210 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if  
72211 the requested action cannot be performed on a file or directory encountered in the  
72212 hierarchy, the utility shall issue a diagnostic message to standard error and  
72213 continue processing the remaining files in the hierarchy, but the final exit status  
72214 shall be returned as non-zero.

- 72215 • If the requested action characterized by an option or option-argument cannot be  
72216 performed, the utility shall issue a diagnostic message to standard error and the  
72217 exit status returned shall be non-zero.

72218 • When an unrecoverable error condition is encountered, the utility shall exit with a  
72219 non-zero exit status.

72220 • A diagnostic message shall be written to standard error whenever an error  
72221 condition occurs.

72222 When a utility encounters an error condition several actions are possible, depending on  
72223 the severity of the error and the state of the utility. Included in the possible actions of  
72224 various utilities are: deletion of temporary or intermediate work files; deletion of  
72225 incomplete files; validity checking of the file system or directory.

72226 **Default Behavior:** When this section is listed as “Default.”, it means that any changes  
72227 to the environment are unspecified.

#### 72228 APPLICATION USAGE

72229 This section is informative.

72230 The APPLICATION USAGE section gives advice to the application programmer or  
72231 user about the way the utility should be used.

#### 72232 EXAMPLES

72233 This section is informative.

72234 The EXAMPLES section gives one or more examples of usage, where appropriate. In  
72235 the event of conflict between an example and a normative part of the specification, the  
72236 normative material is to be taken as correct.

72237 In all examples, quoting has been used, showing how sample commands (utility names  
72238 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to  
72239 the *system()* function defined in the System Interfaces volume of POSIX.1-2008. Such  
72240 quoting would not be used if the utility is invoked using one of the *exec* functions  
72241 defined in the System Interfaces volume of POSIX.1-2008.

#### 72242 RATIONALE

72243 This section is informative.

72244 This section contains historical information concerning the contents of this volume of  
72245 POSIX.1-2008 and why features were included or discarded by the standard  
72246 developers.

#### 72247 FUTURE DIRECTIONS

72248 This section is informative.

72249 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there  
72250 is not necessarily a commitment to implement all of these future directions in their  
72251 entirety.

#### 72252 SEE ALSO

72253 This section is informative.

72254 The SEE ALSO section lists related entries.

#### 72255 CHANGE HISTORY

72256 This section is informative.

72257 This section shows the derivation of the entry and any significant changes that have  
72258 been made to it.

72259 Certain of the standard utilities describe how they can invoke other utilities or applications, such  
72260 as by passing a command string to the command interpreter. The external influences (STDIN,  
72261 ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF

72262 ERRORS, and so on) of such invoked utilities are not described in the section concerning the  
72263 standard utility that invokes them.

## 72264 1.5 Considerations for Utilities in Support of Files of Arbitrary Size

72265 The following utilities support files of any size up to the maximum that can be created by the  
72266 implementation. This support includes correct writing of file size-related values (such as file  
72267 sizes and offsets, line numbers, and block counts) and correct interpretation of command line  
72268 arguments that contain such values.

72269	<i>basename</i>	Return non-directory portion of pathname.
72270	<i>cat</i>	Concatenate and print files.
72271	<i>cd</i>	Change working directory.
72272	<i>chgrp</i>	Change file group ownership.
72273	<i>chmod</i>	Change file modes.
72274	<i>chown</i>	Change file ownership.
72275	<i>cksum</i>	Write file checksums and sizes.
72276	<i>cmp</i>	Compare two files.
72277	<i>cp</i>	Copy files.
72278	<i>dd</i>	Convert and copy a file.
72279	<i>df</i>	Report free disk space.
72280	<i>dirname</i>	Return directory portion of pathname.
72281	<i>du</i>	Estimate file space usage.
72282	<i>find</i>	Find files.
72283	<i>ln</i>	Link files.
72284	<i>ls</i>	List directory contents.
72285	<i>mkdir</i>	Make directories.
72286	<i>mv</i>	Move files.
72287	<i>pathchk</i>	Check pathnames.
72288	<i>pwd</i>	Return working directory name.
72289	<i>rmdir</i>	Remove directory entries.
72290	<i>rmdir</i>	Remove directories.
72291	<i>sh</i>	Shell, the standard command language interpreter.
72292	<i>sum</i>	Print checksum and block or byte count of a file.
72293	<i>test</i>	Evaluate expression.
72294	<i>touch</i>	Change file access and modification times.
72295	<i>ulimit</i>	Set or report file size limit.

72296 Exceptions to the requirement that utilities support files of any size up to the maximum are as  
72297 follows:

- 72298 1. Uses of files as command scripts, or for configuration or control, are exempt. For example,  
72299 it is not required that *sh* be able to read an arbitrarily large **.profile**.
- 72300 2. Shell input and output redirection are exempt. For example, it is not required that the  
72301 redirections *sum < file* or *echo foo > file* succeed for an arbitrarily large existing file.

72302 **1.6 Built-In Utilities**

72303 Any of the standard utilities may be implemented as regular built-in utilities within the  
72304 command language interpreter. This is usually done to increase the performance of frequently  
72305 used utilities or to achieve functionality that would be more difficult in a separate environment.  
72306 The utilities named in [Table 1-5](#) are frequently provided in built-in form. All of the utilities  
72307 named in the table have special properties in terms of command search order within the shell, as  
72308 described in [Section 2.9.1.1](#) (on page 2317).

72309 **Table 1-5** Regular Built-In Utilities

72310	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
72311	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
72312	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
72313	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

72314 However, all of the standard utilities, including the regular built-ins in the table, but not the  
72315 special built-ins described in [Section 2.14](#) (on page 2334), shall be implemented in a manner so  
72316 that they can be accessed via the *exec* family of functions as defined in the System Interfaces  
72317 volume of POSIX.1-2008 and can be invoked directly by those standard utilities that require it  
72318 (*env, find, nice, nohup, time, xargs*).

72319

Chapter 2

72320

## Shell Command Language

72321

This chapter contains the definition of the Shell Command Language.

### 72322 2.1 Shell Introduction

72323

72324

72325

The shell is a command language interpreter. This chapter describes the syntax of that command language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2008.

72326

72327

The shell operates according to the following general overview of operations. The specific details are included in the cited sections of this chapter.

72328

72329

72330

72331

1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2008. If the first line of a file of shell commands starts with the characters "#!", the results are unspecified.

72332

72333

2. The shell breaks the input into tokens: words and operators; see [Section 2.3](#) (on page 2299).

72334

72335

3. The shell parses the input into simple commands (see [Section 2.9.1](#), on page 2316) and compound commands (see [Section 2.9.4](#), on page 2321).

72336

72337

72338

4. The shell performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments; see [Section 2.6](#) (on page 2305).

72339

72340

5. The shell performs redirection (see [Section 2.7](#), on page 2312) and removes redirection operators and their operands from the parameter list.

72341

72342

72343

72344

72345

6. The shell executes a function (see [Section 2.9.5](#), on page 2324), built-in (see [Section 2.14](#), on page 2334), executable file, or script, giving the names of the arguments as positional parameters numbered 1 to *n*, and the name of the command (or in the case of a function within a script, the name of the script) as the positional parameter numbered 0 (see [Section 2.9.1.1](#), on page 2317).

72346

72347

7. The shell optionally waits for the command to complete and collects the exit status (see [Section 2.8.2](#), on page 2315).

72348 **2.2 Quoting**

72349 Quoting is used to remove the special meaning of certain characters or words to the shell.  
 72350 Quoting can be used to preserve the literal meaning of the special characters in the next  
 72351 paragraph, prevent reserved words from being recognized as such, and prevent parameter  
 72352 expansion and command substitution within here-document processing (see [Section 2.7.4](#), on  
 72353 page 2313).

72354 The application shall quote the following characters if they are to represent themselves:

72355 | & ; < > ( ) \$ ` \ " ' <space> <tab> <newline>

72356 and the following may need to be quoted under certain circumstances. That is, these characters  
 72357 may be special depending on conditions described elsewhere in this volume of POSIX.1-2008:

72358 \* ? [ # ~ = %

72359 The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The  
 72360 here-document represents another form of quoting; see [Section 2.7.4](#) (on page 2313).

72361 **2.2.1 Escape Character (Backslash)**

72362 A <backslash> that is not quoted shall preserve the literal value of the following character, with  
 72363 the exception of a <newline>. If a <newline> follows the <backslash>, the shell shall interpret  
 72364 this as line continuation. The <backslash> and <newline> shall be removed before splitting the  
 72365 input into tokens. Since the escaped <newline> is removed entirely from the input and is not  
 72366 replaced by any white space, it cannot serve as a token separator.

72367 **2.2.2 Single-Quotes**

72368 Enclosing characters in single-quotes ( ' ' ) shall preserve the literal value of each character  
 72369 within the single-quotes. A single-quote cannot occur within single-quotes.

72370 **2.2.3 Double-Quotes**

72371 Enclosing characters in double-quotes ( " " ) shall preserve the literal value of all characters  
 72372 within the double-quotes, with the exception of the characters backquote, <dollar-sign>, and  
 72373 <backslash>, as follows:

72374 \$ The <dollar-sign> shall retain its special meaning introducing parameter expansion (see  
 72375 [Section 2.6.2](#), on page 2306), a form of command substitution (see [Section 2.6.3](#), on page  
 72376 2309), and arithmetic expansion (see [Section 2.6.4](#), on page 2310).

72377 The input characters within the quoted string that are also enclosed between "\$ (" and the  
 72378 matching ') ' shall not be affected by the double-quotes, but rather shall define that  
 72379 command whose output replaces the "\$ (...)" when the word is expanded. The  
 72380 tokenizing rules in [Section 2.3](#) (on page 2299), not including the alias substitutions in  
 72381 [Section 2.3.1](#) (on page 2300), shall be applied recursively to find the matching ') '.

72382 Within the string of characters from an enclosed "\$ {" to the matching ' } ' , an even number  
 72383 of unescaped double-quotes or single-quotes, if any, shall occur. A preceding <backslash>  
 72384 character shall be used to escape a literal ' ' or ' } ' . The rule in [Section 2.6.2](#) (on page  
 72385 2306) shall be used to determine the matching ' } ' .

72386       ` The backquote shall retain its special meaning introducing the other form of command  
72387 substitution (see [Section 2.6.3](#), on page 2309). The portion of the quoted string from the  
72388 initial backquote and the characters up to the next backquote that is not preceded by a  
72389 <backslash>, having escape characters removed, defines that command whose output  
72390 replaces "`...`" when the word is expanded. Either of the following cases produces  
72391 undefined results:

- 72392           • A single-quoted or double-quoted string that begins, but does not end, within the  
72393           " `...`" sequence
- 72394           • A "`...`" sequence that begins, but does not end, within the same double-quoted  
72395           string

72396       \  
72397       The <backslash> shall retain its special meaning as an escape character (see [Section 2.2.1](#), on  
page 2298) only when followed by one of the following characters when considered special:

72398       \$   `   "   \  
          <newline>

72399       The application shall ensure that a double-quote is preceded by a <backslash> to be included  
72400 within double-quotes. The parameter '@' has special meaning inside double-quotes and is  
72401 described in [Section 2.5.2](#) (on page 2302).

## 72402 2.3 Token Recognition

72403       The shell shall read its input in terms of lines from a file, from a terminal in the case of an  
72404 interactive shell, or from a string in the case of *sh -c* or *system()*. The input lines can be of  
72405 unlimited length. These lines shall be parsed using two major modes: ordinary token recognition  
72406 and processing of here-documents.

72407       When an **io\_here** token has been recognized by the grammar (see [Section 2.10](#), on page 2325),  
72408 one or more of the subsequent lines immediately following the next **NEWLINE** token form the  
72409 body of one or more here-documents and shall be parsed according to the rules of [Section 2.7.4](#)  
72410 (on page 2313).

72411       When it is not processing an **io\_here**, the shell shall break its input into tokens by applying the  
72412 first applicable rule below to the next character in its input. The token shall be from the current  
72413 position in the input until a token is delimited according to one of the rules below; the characters  
72414 forming the token are exactly those in the input, including any quoting characters. If it is  
72415 indicated that a token is delimited, and no characters have been included in a token, processing  
72416 shall continue until an actual token is delimited.

- 72417           1. If the end of input is recognized, the current token shall be delimited. If there is no  
72418           current token, the end-of-input indicator shall be returned as the token.
- 72419           2. If the previous character was used as part of an operator and the current character is not  
72420           quoted and can be used with the current characters to form an operator, it shall be used as  
72421           part of that (operator) token.
- 72422           3. If the previous character was used as part of an operator and the current character cannot  
72423           be used with the current characters to form an operator, the operator containing the  
72424           previous character shall be delimited.
- 72425           4. If the current character is <backslash>, single-quote, or double-quote and it is not quoted,  
72426           it shall affect quoting for subsequent characters up to the end of the quoted text. The rules  
72427           for quoting are as described in [Section 2.2](#) (on page 2298). During token recognition no  
72428           substitutions shall be actually performed, and the result token shall contain exactly the  
72429           characters that appear in the input (except for <newline> joining), unmodified, including

- 72430 any embedded or enclosing quotes or substitution operators, between the <quotation-
- 72431 mark> and the end of the quoted text. The token shall not be delimited by the end of the
- 72432 quoted field.
- 72433 5. If the current character is an unquoted ' \$ ' or ' ` ', the shell shall identify the start of any
- 72434 candidates for parameter expansion (Section 2.6.2, on page 2306), command substitution
- 72435 (Section 2.6.3, on page 2309), or arithmetic expansion (Section 2.6.4, on page 2310) from
- 72436 their introductory unquoted character sequences: ' \$ ' or "\$ {", "\$ (" or ' ` ', and "\$ ( (",
- 72437 respectively. The shell shall read sufficient input to determine the end of the unit to be
- 72438 expanded (as explained in the cited sections). While processing the characters, if
- 72439 instances of expansions or quoting are found nested within the substitution, the shell
- 72440 shall recursively process them in the manner specified for the construct that is found. The
- 72441 characters found from the beginning of the substitution to its end, allowing for any
- 72442 recursion necessary to recognize embedded constructs, shall be included unmodified in
- 72443 the result token, including any embedded or enclosing substitution operators or quotes.
- 72444 The token shall not be delimited by the end of the substitution.
- 72445 6. If the current character is not quoted and can be used as the first character of a new
- 72446 operator, the current token (if any) shall be delimited. The current character shall be used
- 72447 as the beginning of the next (operator) token.
- 72448 7. If the current character is an unquoted <newline>, the current token shall be delimited.
- 72449 8. If the current character is an unquoted <blank>, any token containing the previous
- 72450 character is delimited and the current character shall be discarded.
- 72451 9. If the previous character was part of a word, the current character shall be appended to
- 72452 that word.
- 72453 10. If the current character is a ' # ', it and all subsequent characters up to, but excluding, the
- 72454 next <newline> shall be discarded as a comment. The <newline> that ends the line is not
- 72455 considered part of the comment.
- 72456 11. The current character is used as the start of a new word.

72457 Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page

72458 2325).

### 72459 2.3.1 Alias Substitution

72460 After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on

72461 page 2325), a resulting word that is identified to be the command name word of a simple

72462 command shall be examined to determine whether it is an unquoted, valid alias name. However,

72463 reserved words in correct grammatical context shall not be candidates for alias substitution. A

72464 valid alias name (see XBD Section 3.10, on page 34) shall be one that has been defined by the

72465 *alias* utility and not subsequently undefined using *unalias*. Implementations also may provide

72466 predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in

72467 recursive aliasing, if the shell is not currently processing an alias of the same name, the word

72468 shall be replaced by the value of the alias; otherwise, it shall not be replaced.

72469 If the value of the alias replacing the word ends in a <blank>, the shell shall check the next

72470 command word for alias substitution; this process shall continue until a word is found that is

72471 not a valid alias or an alias value does not end in a <blank>.

72472 When used as specified by this volume of POSIX.1-2008, alias definitions shall not be inherited

72473 by separate invocations of the shell or by the utility execution environments invoked by the

72474 shell; see Section 2.12 (on page 2331).

## 72475 2.4 Reserved Words

72476 Reserved words are words that have special meaning to the shell; see [Section 2.9](#) (on page 2316).  
72477 The following words shall be recognized as reserved words:

72478	<b>!</b>	<b>do</b>	<b>esac</b>	<b>in</b>
72479	<b>{</b>	<b>done</b>	<b>fi</b>	<b>then</b>
72480	<b>}</b>	<b>elif</b>	<b>for</b>	<b>until</b>
72481	<b>case</b>	<b>else</b>	<b>if</b>	<b>while</b>

72482 This recognition shall only occur when none of the characters is quoted and when the word is  
72483 used as:

- 72484 • The first word of a command
- 72485 • The first word following one of the reserved words other than **case**, **for**, or **in**
- 72486 • The third word in a **case** command (only **in** is valid in this case)
- 72487 • The third word in a **for** command (only **in** and **do** are valid in this case)

72488 See the grammar in [Section 2.10](#) (on page 2325).

72489 The following words may be recognized as reserved words on some implementations (when  
72490 none of the characters are quoted), causing unspecified results:

72491	<b>[[</b>	<b>]]</b>	<b>function</b>	<b>select</b>
-------	-----------	-----------	-----------------	---------------

72492 Words that are the concatenation of a name and a <colon> ( ' : ' ) are reserved; their use produces  
72493 unspecified results.

## 72494 2.5 Parameters and Variables

72495 A parameter can be denoted by a name, a number, or one of the special characters listed in  
72496 [Section 2.5.2](#) (on page 2302). A variable is a parameter denoted by a name.

72497 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can  
72498 only be unset by using the *unset* special built-in command.

### 72499 2.5.1 Positional Parameters

72500 A positional parameter is a parameter denoted by the decimal value represented by one or more  
72501 digits, other than the single digit 0. The digits denoting the positional parameters shall always  
72502 be interpreted as a decimal value, even if there is a leading zero. When a positional parameter  
72503 with more than one digit is specified, the application shall enclose the digits in braces (see  
72504 [Section 2.6.2](#), on page 2306). Positional parameters are initially assigned when the shell is  
72505 invoked (see *sh*), temporarily replaced when a shell function is invoked (see [Section 2.9.5](#), on  
72506 page 2324), and can be reassigned with the *set* special built-in command.

72507 **2.5.2 Special Parameters**

72508 Listed below are the special parameters and the values to which they shall expand. Only the  
72509 values of the special parameters are listed; see [Section 2.6](#) (on page 2305) for a detailed summary  
72510 of all the stages involved in expanding words.

72511 @ Expands to the positional parameters, starting from one. When the expansion occurs within  
72512 double-quotes, and where field splitting (see [Section 2.6.5](#), on page 2311) is performed, each  
72513 positional parameter shall expand as a separate field, with the provision that the expansion  
72514 of the first parameter shall still be joined with the beginning part of the original word  
72515 (assuming that the expanded parameter was embedded within a word), and the expansion  
72516 of the last parameter shall still be joined with the last part of the original word. If there are  
72517 no positional parameters, the expansion of ' @ ' shall generate zero fields, even when ' @ ' is  
72518 double-quoted.

72519 \* Expands to the positional parameters, starting from one. When the expansion occurs within  
72520 a double-quoted string (see [Section 2.2.3](#), on page 2298), it shall expand to a single field with  
72521 the value of each parameter separated by the first character of the *IFS* variable, or by a  
72522 <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its  
72523 first character does not exist, so the parameter values are concatenated.

72524 # Expands to the decimal number of positional parameters. The command name (parameter  
72525 0) shall not be counted in the number given by ' # ' because it is a special parameter, not a  
72526 positional parameter.

72527 ? Expands to the decimal exit status of the most recent pipeline (see [Section 2.9.2](#), on page  
72528 2318).

72529 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated  
72530 into a string) as specified on invocation, by the *set* special built-in command, or implicitly  
72531 by the shell.

72532 \$ Expands to the decimal process ID of the invoked shell. In a subshell (see [Section 2.12](#), on  
72533 page 2331), ' \$ ' shall expand to the same value as that of the current shell.

72534 ! Expands to the decimal process ID of the most recent background command (see [Section  
72535 2.9.3](#), on page 2319) executed from the current shell. (For example, background commands  
72536 executed from subshells do not affect the value of " \$ ! " in the current shell environment.)  
72537 For a pipeline, the process ID is that of the last command in the pipeline.

72538 0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3163) for a detailed  
72539 description of how this name is derived.

72540 See the description of the *IFS* variable in [Section 2.5.3](#).

72541 **2.5.3 Shell Variables**

72542 Variables shall be initialized from the environment (as defined by XBD [Chapter 8](#) (on page 173)  
72543 and the *exec* function in the System Interfaces volume of POSIX.1-2008) and can be given new  
72544 values with variable assignment commands. If a variable is initialized from the environment, it  
72545 shall be marked for export immediately; see the *export* special built-in. New variables can be  
72546 defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name*  
72547 parameter in a **for** loop, with the  $\${name=word}$  expansion, or with other mechanisms provided  
72548 as implementation extensions.

72549		The following variables shall affect the execution of the shell:
72550	UP XSI	<b>ENV</b> The processing of the <i>ENV</i> shell variable shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option.
72551		
72552		
72553		This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2, on page 2306) by the shell and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of <i>ENV</i> is not an absolute pathname, the results are unspecified. <i>ENV</i> shall be ignored if the user's real and effective user IDs or real and effective group IDs are different.
72554		
72555		
72556		
72557		
72558		
72559		
72560	<i>HOME</i>	The pathname of the user's home directory. The contents of <i>HOME</i> are used in tilde expansion (see Section 2.6.1, on page 2305).
72561		
72562	<i>IFS</i>	A string treated as a list of characters that is used for field splitting and to split lines into fields with the <i>read</i> command.
72563		
72564		If <i>IFS</i> is not set, it shall behave as normal for an unset variable, except that field splitting by the shell and line splitting by the <i>read</i> command shall be performed as if the value of <i>IFS</i> is <space><tab><newline>; see Section 2.6.5 (on page 2311).
72565		
72566		
72567		
72568		Implementations may ignore the value of <i>IFS</i> in the environment, or the absence of <i>IFS</i> from the environment, at the time the shell is invoked, in which case the shell shall set <i>IFS</i> to <space><tab><newline> when it is invoked.
72569		
72570		
72571	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
72572		
72573		
72574		
72575	<i>LC_ALL</i>	The value of this variable overrides the <i>LC_*</i> variables and <i>LANG</i> , as described in XBD Chapter 8 (on page 173).
72576		
72577	<i>LC_COLLATE</i>	Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
72578		
72579	<i>LC_CTYPE</i>	Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class <b>alpha</b> ) and <blank> characters (character class <b>blank</b> ), and the behavior of character classes within pattern matching. Changing the value of <i>LC_CTYPE</i> after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells. Invoking a shell script or performing <i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .
72580		
72581		
72582		
72583		
72584		
72585		
72586		
72587	<i>LC_MESSAGES</i>	Determine the language in which messages should be written.
72588	<i>LINENO</i>	Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets <i>LINENO</i> , the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of <i>LINENO</i> is unspecified. This volume of POSIX.1-2008 specifies the effects of the variable only for systems supporting the User Portability Utilities option.
72589		
72590		
72591		
72592		
72593		
72594		

72595	XSI	<b>NLSPATH</b>	Determine the location of message catalogs for the processing of
72596			<b>LC_MESSAGES.</b>
72597		<b>PATH</b>	A string formatted as described in XBD Chapter 8 (on page 173), used to effect
72598			command interpretation; see Section 2.9.1.1 (on page 2317).
72599		<b>PPID</b>	Set by the shell to the decimal value of its parent process ID during
72600			initialization of the shell. In a subshell (see Section 2.12, on page 2331), <i>PPID</i>
72601			shall be set to the same value as that of the parent of the current shell. For
72602			example, <i>echo \$PPID</i> and ( <i>echo \$PPID</i> ) would produce the same value. This
72603			volume of POSIX.1-2008 specifies the effects of the variable only for systems
72604			supporting the User Portability Utilities option.
72605		<b>PS1</b>	Each time an interactive shell is ready to read a command, the value of this
72606			variable shall be subjected to parameter expansion and written to standard
72607			error. The default value shall be "\$ ". For users who have specific additional
72608			implementation-defined privileges, the default may be another,
72609			implementation-defined value. The shell shall replace each instance of the
72610			character '!' in <i>PS1</i> with the history file number of the next command to be
72611			typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal
72612			character '!' in the prompt. This volume of POSIX.1-2008 specifies the effects
72613			of the variable only for systems supporting the User Portability Utilities
72614			option.
72615		<b>PS2</b>	Each time the user enters a <newline> prior to completing a command line in
72616			an interactive shell, the value of this variable shall be subjected to parameter
72617			expansion and written to standard error. The default value is "> ". This
72618			volume of POSIX.1-2008 specifies the effects of the variable only for systems
72619			supporting the User Portability Utilities option.
72620		<b>PS4</b>	When an execution trace ( <i>set -x</i> ) is being performed in an interactive shell,
72621			before each line in the execution trace, the value of this variable shall be
72622			subjected to parameter expansion and written to standard error. The default
72623			value is "+ ". This volume of POSIX.1-2008 specifies the effects of the
72624			variable only for systems supporting the User Portability Utilities option.
72625		<b>PWD</b>	Set by the shell and by the <i>cd</i> utility. In the shell the value shall be initialized
72626			from the environment as follows. If a value for <i>PWD</i> is passed to the shell in
72627			the environment when it is executed, the value is an absolute pathname of the
72628			current working directory that is no longer than {PATH_MAX} bytes including
72629			the terminating null byte, and the value does not contain any components that
72630			are dot or dot-dot, then the shell shall set <i>PWD</i> to the value from the
72631			environment. Otherwise, if a value for <i>PWD</i> is passed to the shell in the
72632			environment when it is executed, the value is an absolute pathname of the
72633			current working directory, and the value does not contain any components
72634			that are dot or dot-dot, then it is unspecified whether the shell sets <i>PWD</i> to the
72635			value from the environment or sets <i>PWD</i> to the pathname that would be
72636			output by <i>pwd -P</i> . Otherwise, the <i>sh</i> utility sets <i>PWD</i> to the pathname that
72637			would be output by <i>pwd -P</i> . In cases where <i>PWD</i> is set to the value from the
72638			environment, the value can contain components that refer to files of type
72639			symbolic link. In cases where <i>PWD</i> is set to the pathname that would be
72640			output by <i>pwd -P</i> , if there is insufficient permission on the current working
72641			directory, or on any parent of that directory, to determine what that pathname
72642			would be, the value of <i>PWD</i> is unspecified. Assignments to this variable may
72643			be ignored. If an application sets or unsets the value of <i>PWD</i> , the behaviors of
72644			the <i>cd</i> and <i>pwd</i> utilities are unspecified.

## 2.6 Word Expansions

This section describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following sections.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word expand to a single field. It is only field splitting or pathname expansion that can create multiple fields from a single word. The single exception to this rule is the expansion of the special parameter '@' within double-quotes, as described in [Section 2.5.2](#) (on page 2302).

The order of word expansion shall be as follows:

1. Tilde expansion (see [Section 2.6.1](#)), parameter expansion (see [Section 2.6.2](#), on page 2306), command substitution (see [Section 2.6.3](#), on page 2309), and arithmetic expansion (see [Section 2.6.4](#), on page 2310) shall be performed, beginning to end. See item 5 in [Section 2.3](#) (on page 2299).
2. Field splitting (see [Section 2.6.5](#), on page 2311) shall be performed on the portions of the fields generated by step 1, unless *IFS* is null.
3. Pathname expansion (see [Section 2.6.6](#), on page 2311) shall be performed, unless *set -f* is in effect.
4. Quote removal (see [Section 2.6.7](#), on page 2311) shall always be performed last.

The expansions described in this section shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The '\$' character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric, the name of one of the special parameters (see [Section 2.5.2](#), on page 2302), a valid first character of a variable name, a <left-curly-bracket> ('{') or a <left-parenthesis>, the result is unspecified.

### 2.6.1 Tilde Expansion

A "tilde-prefix" consists of an unquoted <tilde> character at the beginning of a word, followed by all of the characters preceding the first unquoted <slash> in the word, or all the characters in the word if there is no <slash>. In an assignment (see [XBD Section 4.22](#), on page 118), multiple tilde-prefixes can be used: at the beginning of the word (that is, following the <equals-sign> of the assignment), following any unquoted <colon>, or both. A tilde-prefix in an assignment is terminated by the first unquoted <colon> or <slash>. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the <tilde> are treated as a possible login name from the user database. A portable login name cannot contain characters outside the set given in the description of the *LOGNAME* environment variable in [XBD Section 8.3](#) (on page 177). If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name obtained using the *getpwnam()* function as defined in the System Interfaces volume of POSIX.1-2008. If the system does not recognize the login name, the results are undefined.

The pathname resulting from tilde expansion shall be treated as if quoted to prevent it being

72689 altered by field splitting and pathname expansion.

## 72690 2.6.2 Parameter Expansion

72691 The format for parameter expansion is as follows:

72692  $\${expression}$

72693 where *expression* consists of all characters until the matching `'``'`. Any `'``'` escaped by a  
72694 `<backslash>` or within a quoted string, and characters in embedded arithmetic expansions,  
72695 command substitutions, and variable expansions, shall not be examined in determining the  
72696 matching `'``'`.

72697 The simplest form for parameter expansion is:

72698  $\${parameter}$

72699 The value, if any, of *parameter* shall be substituted.

72700 The parameter name or symbol can be enclosed in braces, which are optional except for  
72701 positional parameters with more than one digit or when *parameter* is followed by a character that  
72702 could be interpreted as part of the name. The matching closing brace shall be determined by  
72703 counting brace levels, skipping over enclosed quoted strings, and command substitutions.

72704 If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest  
72705 valid name (see XBD Section 3.230, on page 70), whether or not the symbol represented by that  
72706 name exists.

72707 If a parameter expansion occurs inside double quotes:

- 72708 • Pathname expansion shall not be performed on the results of the expansion.
- 72709 • Field splitting shall not be performed on the results of the expansion, with the exception of  
72710 `'@'`; see Section 2.5.2 (on page 2302).

72711 In addition, a parameter expansion can be modified by using one of the following formats. In  
72712 each case that a value of *word* is needed (based on the state of *parameter*, as described below),  
72713 *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and  
72714 arithmetic expansion. If *word* is not needed, it shall not be expanded. The `'``'` character that  
72715 delimits the following parameter expansion modifications shall be determined as described  
72716 previously in this section and in Section 2.2.3 (on page 2298). (For example,  $\{\{\mathbf{foo-bar}\}\mathbf{xyz}\}$   
72717 would result in the expansion of **foo** followed by the string **xyz** if **foo** is set, else the string  
72718 `"barxyz"`).

72719  $\${parameter}:-word$  **Use Default Values.** If *parameter* is unset or null, the expansion of *word*  
72720 shall be substituted; otherwise, the value of *parameter* shall be substituted.

72721  $\${parameter}:=word$  **Assign Default Values.** If *parameter* is unset or null, the expansion of  
72722 *word* shall be assigned to *parameter*. In all cases, the final value of  
72723 *parameter* shall be substituted. Only variables, not positional parameters  
72724 or special parameters, can be assigned in this way.

72725  $\${parameter}?:[word]$  **Indicate Error if Null or Unset.** If *parameter* is unset or null, the  
72726 expansion of *word* (or a message indicating it is unset if *word* is omitted)  
72727 shall be written to standard error and the shell exits with a non-zero exit  
72728 status. Otherwise, the value of *parameter* shall be substituted. An  
72729 interactive shell need not exit.



72768 **Examples**72769 `${parameter:-word}`72770 In this example, `ls` is executed only if `x` is null or unset. (The `$(ls)` command substitution notation is explained in [Section 2.6.3](#) (on page 2309).)72771 `echo ${x:-$(ls)}`72772 `unset X`72773 `echo ${X:=abc}`72774 `abc`72775 `unset posix`72776 `echo ${posix:?}`72777 `sh: posix: parameter null or not set`72778 `set a b c`72779 `echo ${3:+posix}`72780 `posix`72781 `HOME=/usr/posix`72782 `echo ${#HOME}`72783 `10`72784 `x=file.c`72785 `echo ${x%.c}.o`72786 `file.o`72787 `x=posix/src/std`72788 `echo ${x%*/}`72789 `posix`72790 `x=$HOME/src/cmd`72791 `echo ${x#HOME}`72792 `/src/cmd`72793 `x=/one/two/three`72794 `echo ${x##*/}`72795 `three`72796 `x=$HOME/src/cmd`72797 `echo ${x#HOME}`72798 `/src/cmd`72799 `x=/one/two/three`72800 `echo ${x##*/}`72801 `three`

72802

72803

72804

72805

72806

72807

The double-quoting of patterns is different depending on where the double-quotes are placed:

`"${x#*}"` The <asterisk> is a pattern character.

`${x#"*"}`  The literal <asterisk> is quoted and not special.

### 72808 2.6.3 Command Substitution

72809 Command substitution allows the output of a command to be substituted in place of the  
72810 command name itself. Command substitution shall occur when the command is enclosed as  
72811 follows:

72812 `$(command)`

72813 or (backquoted version):

72814 ``command``

72815 The shell shall expand the command substitution by executing *command* in a subshell  
72816 environment (see Section 2.12, on page 2331) and replacing the command substitution (the text  
72817 of *command* plus the enclosing "\$ ( )" or backquotes) with the standard output of the command,  
72818 removing sequences of one or more <newline> characters at the end of the substitution.  
72819 Embedded <newline> characters before the end of the output shall not be removed; however,  
72820 they may be treated as field delimiters and eliminated during field splitting, depending on the  
72821 value of *IFS* and quoting that is in effect. If the output contains any null bytes, the behavior is  
72822 unspecified.

72823 Within the backquoted style of command substitution, <backslash> shall retain its literal  
72824 meaning, except when followed by: '\$', '`', or <backslash>. The search for the matching  
72825 backquote shall be satisfied by the first unquoted non-escaped backquote; during this search, if a  
72826 non-escaped backquote is encountered within a shell comment, a here-document, an embedded  
72827 command substitution of the \$(*command*) form, or a quoted string, undefined results occur. A  
72828 single-quoted or double-quoted string that begins, but does not end, within the "`...`"  
72829 sequence produces undefined results.

72830 With the \$(*command*) form, all characters following the open parenthesis to the matching closing  
72831 parenthesis constitute the *command*. Any valid shell script can be used for *command*, except a  
72832 script consisting solely of redirections which produces unspecified results.

72833 The results of command substitution shall not be processed for further tilde expansion,  
72834 parameter expansion, command substitution, or arithmetic expansion. If a command  
72835 substitution occurs inside double-quotes, field splitting and pathname expansion shall not be  
72836 performed on the results of the substitution.

72837 Command substitution can be nested. To specify nesting within the backquoted version, the  
72838 application shall precede the inner backquotes with <backslash> characters; for example:

72839 `\`command\``

72840 If the command substitution consists of a single subshell, such as:

72841 `$( (command) )`

72842 a conforming application shall separate the "\$ (" and ' (' into two tokens (that is, separate  
72843 them with white space). This is required to avoid any ambiguities with arithmetic expansion.

72844 **2.6.4 Arithmetic Expansion**

72845 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and  
 72846 substituting its value. The format for arithmetic expansion shall be as follows:

72847 `$( (expression) )`

72848 The expression shall be treated as if it were in double-quotes, except that a double-quote inside  
 72849 the expression is not treated specially. The shell shall expand all tokens in the expression for  
 72850 parameter expansion, command substitution, and quote removal.

72851 Next, the shell shall treat this as an arithmetic expression and substitute the value of the  
 72852 expression. The arithmetic expression shall be processed according to the rules given in [Section](#)  
 72853 [1.1.2.1](#) (on page 2283), with the following exceptions:

- 72854 • Only signed long integer arithmetic is required.
- 72855 • Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in  
 72856 the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- 72857 • The `sizeof()` operator and the prefix and postfix "++" and "--" operators are not required.
- 72858 • Selection, iteration, and jump statements are not supported.

72859 All changes to variables in an arithmetic expression shall be in effect after the arithmetic  
 72860 expansion, as in the parameter expansion " `${x=value} "`"

72861 If the shell variable `x` contains a value that forms a valid integer constant, then the arithmetic  
 72862 expansions " `$( (x) ) "`" and " `$( ($x) ) "`" shall return the same value.

72863 As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell  
 72864 may use a signed integer type with a rank larger than the rank of **signed long**. The shell may  
 72865 use a real-floating type instead of **signed long** as long as it does not affect the results in cases  
 72866 where there is no overflow. If the expression is invalid, the expansion fails and the shell shall  
 72867 write a message to standard error indicating the failure.

72868 **Examples**

72869 A simple example using arithmetic expansion:

```
72870 # repeat a command 100 times
72871 x=100
72872 while [ $x -gt 0 ]
72873 do
72874     command
72875     x=$((x-1))
72876 done
```

72877 **2.6.5 Field Splitting**

72878 After parameter expansion (Section 2.6.2, on page 2306), command substitution (Section 2.6.3, on  
72879 page 2309), and arithmetic expansion (Section 2.6.4, on page 2310), the shell shall scan the results  
72880 of expansions and substitutions that did not occur in double-quotes for field splitting and  
72881 multiple fields can result.

72882 The shell shall treat each character of the *IFS* as a delimiter and use the delimiters as field  
72883 terminators to split the results of parameter expansion and command substitution into fields.

72884 1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of  
72885 <space>, <tab>, or <newline> characters at the beginning or end of the input shall be  
72886 ignored and any sequence of those characters within the input shall delimit a field. For  
72887 example, the input:

72888 <newline><space><tab>foo<tab><tab>bar<space>

72889 yields two fields, **foo** and **bar**.

72890 2. If the value of *IFS* is null, no field splitting shall be performed.

72891 3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space”  
72892 is used to mean any sequence (zero or more instances) of white-space characters that are  
72893 in the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of  
72894 <space> and <tab> characters is considered *IFS* white space).

72895 a. *IFS* white space shall be ignored at the beginning and end of the input.

72896 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along  
72897 with any adjacent *IFS* white space, shall delimit a field, as described previously.

72898 c. Non-zero-length *IFS* white space shall delimit a field.

72899 **2.6.6 Pathname Expansion**

72900 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be  
72901 expanded using the algorithm described in Section 2.13 (on page 2332), qualified by the rules in  
72902 Section 2.13.3 (on page 2333).

72903 **2.6.7 Quote Removal**

72904 The quote characters (<backslash>, single-quote, and double-quote) that were present in the  
72905 original word shall be removed unless they have themselves been quoted.

72906 **2.7 Redirection**

72907 Redirection is used to open and close files for the current shell execution environment (see  
72908 [Section 2.12](#), on page 2331) or for any command. Redirection operators can be used with  
72909 numbers representing file descriptors (see [XBD Section 3.166](#), on page 60) as described below.

72910 The overall format used for redirection is:

72911 `[n] redir-op word`

72912 The number *n* is an optional decimal number designating the file descriptor number; the  
72913 application shall ensure it is delimited from any preceding text and immediately precede the  
72914 redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the  
72915 redirection expression. For example:

72916 `echo \2>a`

72917 writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is  
72918 recognized. For example:

72919 `echo 2\>a`

72920 writes the characters 2>a to standard output. The optional number, redirection operator, and  
72921 *word* shall not appear in the arguments provided to the command to be executed (if any).

72922 Open files are represented by decimal numbers starting with zero. The largest possible value is  
72923 implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for  
72924 use by the application. These numbers are called "file descriptors". The values 0, 1, and 2 have  
72925 special meaning and conventional uses and are implied by certain redirection operations; they  
72926 are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs  
72927 usually take their input from standard input, and write output on standard output. Error  
72928 messages are usually written on standard error. The redirection operators can be preceded by  
72929 one or more digits (with no intervening <blank> characters allowed) to designate the file  
72930 descriptor number.

72931 If the redirection operator is "<<" or "<<=", the word that follows the redirection operator shall  
72932 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For  
72933 the other redirection operators, the word that follows the redirection operator shall be subjected  
72934 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and  
72935 quote removal. Pathname expansion shall not be performed on the word by a non-interactive  
72936 shell; an interactive shell may perform it, but shall do so only when the expansion would result  
72937 in one word.

72938 If more than one redirection operator is specified with a command, the order of evaluation is  
72939 from beginning to end.

72940 A failure to open or create a file shall cause a redirection to fail.

72941 **2.7.1 Redirecting Input**

72942 Input redirection shall cause the file whose name results from the expansion of *word* to be  
72943 opened for reading on the designated file descriptor, or standard input if the file descriptor is  
72944 not specified.

72945 The general format for redirecting input is:

72946 `[n] <word`

72947 where the optional *n* represents the file descriptor number. If the number is omitted, the

72948 redirection shall refer to standard input (file descriptor 0).

### 72949 2.7.2 Redirecting Output

72950 The two general formats for redirecting output are:

72951 `[n]>word`

72952 `[n]>|word`

72953 where the optional *n* represents the file descriptor number. If the number is omitted, the  
72954 redirection shall refer to standard output (file descriptor 1).

72955 Output redirection using the '`>`' format shall fail if the *noclobber* option is set (see the  
72956 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.  
72957 Otherwise, redirection using the '`>`' or '`>|`' formats shall cause the file whose name results  
72958 from the expansion of *word* to be created and opened for output on the designated file  
72959 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;  
72960 otherwise, it shall be truncated to be an empty file after being opened.

### 72961 2.7.3 Appending Redirected Output

72962 Appended output redirection shall cause the file whose name results from the expansion of  
72963 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*  
72964 function as defined in the System Interfaces volume of POSIX.1-2008 was called with the  
72965 `O_APPEND` flag. If the file does not exist, it shall be created.

72966 The general format for appending redirected output is as follows:

72967 `[n]>>word`

72968 where the optional *n* represents the file descriptor number. If the number is omitted, the  
72969 redirection refers to standard output (file descriptor 1).

### 72970 2.7.4 Here-Document

72971 The redirection operators "`<<`" and "`<<-`" both allow redirection of lines contained in a shell  
72972 input file, known as a "here-document", to the input of a command.

72973 The here-document shall be treated as a single word that begins after the next <newline> and  
72974 continues until there is a line containing only the delimiter and a <newline>, with no <blank>  
72975 characters in between. Then the next here-document starts, if there is one. The format is as  
72976 follows:

72977 `[n]<<word`

72978 `here-document`

72979 `delimiter`

72980 where the optional *n* represents the file descriptor number. If the number is omitted, the here-  
72981 document refers to standard input (file descriptor 0).

72982 If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on  
72983 *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the  
72984 *word* itself.

72985 If no characters in *word* are quoted, all lines of the here-document shall be expanded for  
72986 parameter expansion, command substitution, and arithmetic expansion. In this case, the

72987 <backslash> in the input behaves as the <backslash> inside double-quotes (see [Section 2.2.3](#), on  
72988 page 2298). However, the double-quote character (' ') shall not be treated specially within a  
72989 here-document, except when the double-quote appears within "\$ ()", "`", or "\${ }".

72990 If the redirection symbol is "<<-", all leading <tab> characters shall be stripped from input lines  
72991 and the line containing the trailing delimiter. If more than one "<<" or "<<-" operator is  
72992 specified on a line, the here-document associated with the first operator shall be supplied first  
72993 by the application and shall be read first by the shell.

72994 When a here-document is read from a terminal device and the shell is interactive, it shall write  
72995 the contents of the variable *PS2*, processed as described in [Section 2.5.3](#) (on page 2302), to  
72996 standard error before reading each line of input until the delimiter has been recognized.

### 72997 Examples

72998 An example of a here-document follows:

```
72999 cat <<eof1; cat <<eof2
73000 Hi,
73001 eof1
73002 Helene.
73003 eof2
```

## 73004 2.7.5 Duplicating an Input File Descriptor

73005 The redirection operator:

```
73006 [n]<&word
```

73007 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one  
73008 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be  
73009 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a  
73010 file descriptor already open for input, a redirection error shall result; see [Section 2.8.1](#) (on page  
73011 2315). If *word* evaluates to '-', file descriptor *n*, or standard input if *n* is not specified, shall be  
73012 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*  
73013 evaluates to something else, the behavior is unspecified.

## 73014 2.7.6 Duplicating an Output File Descriptor

73015 The redirection operator:

```
73016 [n]>&word
```

73017 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to  
73018 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall  
73019 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent  
73020 a file descriptor already open for output, a redirection error shall result; see [Section 2.8.1](#) (on  
73021 page 2315). If *word* evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is  
73022 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*  
73023 evaluates to something else, the behavior is unspecified.

### 73024 2.7.7 Open File Descriptors for Reading and Writing

73025 The redirection operator:

73026 `[n] <>word`

73027 shall cause the file whose name is the expansion of *word* to be opened for both reading and  
73028 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does  
73029 not exist, it shall be created.

## 73030 2.8 Exit Status and Errors

### 73031 2.8.1 Consequences of Shell Errors

73032 For a non-interactive shell, an error condition encountered by a special built-in (see [Section 2.14](#),  
73033 on page 2334) or other type of utility shall cause the shell to write a diagnostic message to  
73034 standard error and exit as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	Shall exit	Shall exit
Utility syntax error (option or operand error)	Shall exit	Shall not exit
Redirection error	Shall exit	Shall not exit
Variable assignment error	Shall exit	Shall not exit
Expansion error	Shall exit	Shall exit
Command not found	N/A	May exit
Dot script not found	Shall exit	N/A

73043 An expansion error is one that occurs when the shell expansions defined in [Section 2.6](#) (on page  
73044 2305) are carried out (for example, "`$(x!y)`", because `'!'` is not a valid operator); an  
73045 implementation may treat these as syntax errors if it is able to detect them during tokenization,  
73046 rather than during expansion.

73047 If any of the errors shown as "shall exit" or "(may) exit" occur in a subshell, the subshell shall  
73048 (respectively may) exit with a non-zero status, but the script containing the subshell shall not  
73049 exit because of the error.

73050 In all of the cases shown in the table, an interactive shell shall write a diagnostic message to  
73051 standard error without exiting.

### 73052 2.8.2 Exit Status for Commands

73053 Each command has an exit status that can influence the behavior of other shell commands. The  
73054 exit status of commands that are not utilities is documented in this section. The exit status of the  
73055 standard utilities is documented in their respective sections.

73056 If a command is not found, the exit status shall be 127. If the command name is found, but it is  
73057 not an executable utility, the exit status shall be 126. Applications that invoke utilities without  
73058 using the shell should use these exit status values to report similar errors.

73059 If a command fails during word expansion or redirection, its exit status shall be greater than  
73060 zero.

73061 Internally, for purposes of deciding whether a command exits with a non-zero exit status, the  
 73062 shell shall recognize the entire status value retrieved for the command by the equivalent of the  
 73063 *wait()* function WEXITSTATUS macro (as defined in the System Interfaces volume of  
 73064 POSIX.1-2008). When reporting the exit status with the special parameter '?', the shell shall  
 73065 report the full eight bits of exit status available. The exit status of a command that terminated  
 73066 because it received a signal shall be reported as greater than 128.

## 73067 2.9 Shell Commands

73068 This section describes the basic structure of shell commands. The following command  
 73069 descriptions each describe a format of the command that is only used to aid the reader in  
 73070 recognizing the command type, and does not formally represent the syntax. Each description  
 73071 discusses the semantics of the command; for a formal definition of the command language,  
 73072 consult Section 2.10 (on page 2325).

73073 A *command* is one of the following:

- 73074 • Simple command (see Section 2.9.1)
- 73075 • Pipeline (see Section 2.9.2, on page 2318)
- 73076 • List compound-list (see Section 2.9.3, on page 2319)
- 73077 • Compound command (see Section 2.9.4, on page 2321)
- 73078 • Function definition (see Section 2.9.5, on page 2324)

73079 Unless otherwise stated, the exit status of a command shall be that of the last simple command  
 73080 executed by the command. There shall be no limit on the size of any shell command other than  
 73081 that imposed by the underlying system (memory constraints, {ARG\_MAX}, and so on).

### 73082 2.9.1 Simple Commands

73083 A “simple command” is a sequence of optional variable assignments and redirections, in any  
 73084 sequence, optionally followed by words and redirections, terminated by a control operator.

73085 When a given simple command is required to be executed (that is, when any conditional  
 73086 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),  
 73087 the following expansions, assignments, and redirections shall all be performed from the  
 73088 beginning of the command text to the end:

- 73089 1. The words that are recognized as variable assignments or redirections according to  
 73090 Section 2.10.2 (on page 2325) are saved for processing in steps 3 and 4.
- 73091 2. The words that are not variable assignments or redirections shall be expanded. If any  
 73092 fields remain following their expansion, the first field shall be considered the command  
 73093 name and remaining fields are the arguments for the command.
- 73094 3. Redirections shall be performed as described in Section 2.7 (on page 2312).
- 73095 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,  
 73096 command substitution, arithmetic expansion, and quote removal prior to assigning the  
 73097 value.

73098 In the preceding list, the order of steps 3 and 4 may be reversed if no command name results  
 73099 from step 2 or if the command name matches the name of a special built-in utility; see Section  
 73100 2.14 (on page 2334).

73101 If no command name results, variable assignments shall affect the current execution  
 73102 environment. Otherwise, the variable assignments shall be exported for the execution  
 73103 environment of the command and shall not affect the current execution environment (except for  
 73104 special built-ins). If any of the variable assignments attempt to assign a value to a read-only  
 73105 variable, a variable assignment error shall occur. See Section 2.8.1 (on page 2315) for the  
 73106 consequences of these errors.

73107 If there is no command name, any redirections shall be performed in a subshell environment; it  
 73108 is unspecified whether this subshell environment is the same one as that used for a command  
 73109 substitution within the command. (To affect the current execution environment, see the *exec*  
 73110 special built-in.) If any of the redirections performed in the current shell execution environment  
 73111 fail, the command shall immediately fail with an exit status greater than zero, and the shell shall  
 73112 write an error message indicating the failure. See Section 2.8.1 (on page 2315) for the  
 73113 consequences of these failures on interactive and non-interactive shells.

73114 If there is a command name, execution shall continue as described in Section 2.9.1.1. If there is  
 73115 no command name, but the command contained a command substitution, the command shall  
 73116 complete with the exit status of the last command substitution performed. Otherwise, the  
 73117 command shall complete with a zero exit status.

#### 73118 2.9.1.1 Command Search and Execution

73119 If a simple command results in a command name and an optional list of arguments, the  
 73120 following actions shall be performed:

- 73121 1. If the command name does not contain any <slash> characters, the first successful step in  
 73122 the following sequence shall occur:
  - 73123 a. If the command name matches the name of a special built-in utility, that special  
 73124 built-in utility shall be invoked.
  - 73125 b. If the command name matches the name of a function known to this shell, the  
 73126 function shall be invoked as described in Section 2.9.5 (on page 2324). If the  
 73127 implementation has provided a standard utility in the form of a function, it shall  
 73128 not be recognized at this point. It shall be invoked in conjunction with the path  
 73129 search in step 1d.
  - 73130 c. If the command name matches the name of a utility listed in the following table,  
 73131 that utility shall be invoked.

73132	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
73133	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
73134	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
73135	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

- 73136 d. Otherwise, the command shall be searched for using the *PATH* environment  
 73137 variable as described in XBD Chapter 8 (on page 173):

- 73138 i. If the search is successful:
  - 73139 a. If the system has implemented the utility as a regular built-in or as a  
 73140 shell function, it shall be invoked at this point in the path search.
  - 73141 b. Otherwise, the shell executes the utility in a separate utility  
 73142 environment (see Section 2.12, on page 2331) with actions equivalent  
 73143 to calling the *execve()* function as defined in the System Interfaces  
 73144 volume of POSIX.1-2008 with the *path* argument set to the pathname  
 73145 resulting from the search, *arg0* set to the command name, and the

- 73146 remaining arguments set to the operands, if any.
- 73147 If the *execve()* function fails due to an error equivalent to the  
73148 [ENOEXEC] error defined in the System Interfaces volume of  
73149 POSIX.1-2008, the shell shall execute a command equivalent to  
73150 having a shell invoked with the pathname resulting from the search  
73151 as its first operand, with any remaining arguments passed to the new  
73152 shell, except that the value of "\$0" in the new shell may be set to the  
73153 command name. If the executable file is not a text file, the shell may  
73154 bypass this command execution. In this case, it shall write an error  
73155 message, and shall return an exit status of 126.
- 73156 Once a utility has been searched for and found (either as a result of this  
73157 specific search or as part of an unspecified shell start-up activity), an  
73158 implementation may remember its location and need not search for the  
73159 utility again unless the *PATH* variable has been the subject of an assignment.  
73160 If the remembered location fails for a subsequent invocation, the shell shall  
73161 repeat the search to find the new location for the utility, if any.
- 73162 ii. If the search is unsuccessful, the command shall fail with an exit status of  
73163 127 and the shell shall write an error message.
- 73164 2. If the command name contains at least one <slash>, the shell shall execute the utility in a  
73165 separate utility environment with actions equivalent to calling the *execve()* function  
73166 defined in the System Interfaces volume of POSIX.1-2008 with the *path* and *arg0*  
73167 arguments set to the command name, and the remaining arguments set to the operands, if  
73168 any.
- 73169 If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell  
73170 shall execute a command equivalent to having a shell invoked with the command name  
73171 as its first operand, with any remaining arguments passed to the new shell. If the  
73172 executable file is not a text file, the shell may bypass this command execution. In this case,  
73173 it shall write an error message and shall return an exit status of 126.

## 73174 2.9.2 Pipelines

73175 A *pipeline* is a sequence of one or more commands separated by the control operator '|'. The  
73176 standard output of all but the last command shall be connected to the standard input of the next  
73177 command.

73178 The format for a pipeline is:

```
73179 [!] command1 [ | command2 ... ]
```

73180 The standard output of *command1* shall be connected to the standard input of *command2*. The  
73181 standard input, standard output, or both of a command shall be considered to be assigned by  
73182 the pipeline before any redirection specified by redirection operators that are part of the  
73183 command (see Section 2.7, on page 2312).

73184 If the pipeline is not in the background (see Section 2.9.3.1, on page 2319), the shell shall wait for  
73185 the last command specified in the pipeline to complete, and may also wait for all commands to  
73186 complete.

73187 **Exit Status**

73188 If the reserved word **!** does not precede the pipeline, the exit status shall be the exit status of the  
 73189 last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the  
 73190 exit status of the last command. That is, if the last command returns zero, the exit status shall be  
 73191 1; if the last command returns greater than zero, the exit status shall be zero.

73192 **2.9.3 Lists**

73193 An *AND-OR list* is a sequence of one or more pipelines separated by the operators "**&&**" and  
 73194 "**||**".

73195 A *list* is a sequence of one or more AND-OR lists separated by the operators '**;**', '**&'** and  
 73196 optionally terminated by '**;**', '**&'**, or **<newline>**.

73197 The operators "**&&**" and "**||**" shall have equal precedence and shall be evaluated with left  
 73198 associativity. For example, both of the following commands write solely **bar** to standard output:

```
73199 false && echo foo || echo bar
73200 true || echo foo && echo bar
```

73201 A '**;**' or **<newline>** terminator shall cause the preceding AND-OR list to be executed  
 73202 sequentially; an '**&'** shall cause asynchronous execution of the preceding AND-OR list.

73203 The term "compound-list" is derived from the grammar in Section 2.10 (on page 2325); it is  
 73204 equivalent to a sequence of *lists*, separated by **<newline>** characters, that can be preceded or  
 73205 followed by an arbitrary number of **<newline>** characters.

73206 **Examples**

73207 The following is an example that illustrates **<newline>** characters in compound-lists:

```
73208 while
73209     # a couple of <newline>s
73210     # a list
73211     date && who || ls; cat file
73212     # a couple of <newline>s
73213     # another list
73214     wc file > output & true
73215 do
73216     # 2 lists
73217     ls
73218     cat file
73219 done
```

73220 **2.9.3.1 Asynchronous Lists**

73221 If a command is terminated by the control operator **<ampersand>** ('**&'**), the shell shall execute  
 73222 the command asynchronously in a subshell. This means that the shell shall not wait for the  
 73223 command to finish before executing the next command.

73224 The format for running a command in the background is:

```
73225 command1 & [command2 & ... ]
```

73226 The standard input for an asynchronous list, before any explicit redirections are performed, shall  
 73227 be considered to be assigned to a file that has the same properties as `/dev/null`. If it is an  
 73228 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall  
 73229 override this activity.

73230 When an element of an asynchronous list (the portion of the list ended by an `&`,  
 73231 such as `command1`, above) is started by the shell, the process ID of the last command in the  
 73232 asynchronous list element shall become known in the current shell execution environment; see  
 73233 [Section 2.12](#) (on page 2331). This process ID shall remain known until:

- 73234 1. The command terminates and the application waits for the process ID.
- 73235 2. Another asynchronous list is invoked before "\$!" (corresponding to the previous  
 73236 asynchronous list) is expanded in the current execution environment.

73237 The implementation need not retain more than the `{CHILD_MAX}` most recent entries in its list  
 73238 of known process IDs in the current shell execution environment.

#### 73239 **Exit Status**

73240 The exit status of an asynchronous list shall be zero.

#### 73241 2.9.3.2 *Sequential Lists*

73242 Commands that are separated by a `<semicolon>` (`' ; '`) shall be executed sequentially.

73243 The format for executing commands sequentially shall be:

73244 `command1 [ ; command2 ] ...`

73245 Each command shall be expanded and executed in the order specified.

#### 73246 **Exit Status**

73247 The exit status of a sequential list shall be the exit status of the last command in the list.

#### 73248 2.9.3.3 *AND Lists*

73249 The control operator `"&&"` denotes an AND list. The format shall be:

73250 `command1 [ && command2 ] ...`

73251 First `command1` shall be executed. If its exit status is zero, `command2` shall be executed, and so on,  
 73252 until a command has a non-zero exit status or there are no more commands left to execute. The  
 73253 commands are expanded only if they are executed.

#### 73254 **Exit Status**

73255 The exit status of an AND list shall be the exit status of the last command that is executed in the  
 73256 list.

#### 73257 2.9.3.4 *OR Lists*

73258 The control operator `"||"` denotes an OR List. The format shall be:

73259 `command1 [ || command2 ] ...`

73260 First, `command1` shall be executed. If its exit status is non-zero, `command2` shall be executed, and  
 73261 so on, until a command has a zero exit status or there are no more commands left to execute.

73262 **Exit Status**

73263 The exit status of an OR list shall be the exit status of the last command that is executed in the  
73264 list.

73265 **2.9.4 Compound Commands**

73266 The shell has several programming constructs that are “compound commands”, which provide  
73267 control flow for commands. Each of these compound commands has a reserved word or control  
73268 operator at the beginning, and a corresponding terminator reserved word or operator at the end.  
73269 In addition, each can be followed by redirections on the same line as the terminator. Each  
73270 redirection shall apply to all the commands within the compound command that do not  
73271 explicitly override that redirection.

73272 **2.9.4.1 Grouping Commands**

73273 The format for grouping commands is as follows:

73274 (*compound-list*) Execute *compound-list* in a subshell environment; see Section 2.12 (on page  
73275 2331). Variable assignments and built-in commands that affect the  
73276 environment shall not remain in effect after the list finishes.

73277 { *compound-list*;} Execute *compound-list* in the current process environment. The semicolon  
73278 shown here is an example of a control operator delimiting the } reserved  
73279 word. Other delimiters are possible, as shown in Section 2.10 (on page  
73280 2325); a <newline> is frequently used.

73281 **Exit Status**

73282 The exit status of a grouping command shall be the exit status of *compound-list*.

73283 **2.9.4.2 The for Loop**

73284 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**  
73285 loop requires that the reserved words **do** and **done** be used to delimit the sequence of  
73286 commands.

73287 The format for the **for** loop is as follows:

```
73288 for name [ in [word ... ] ]
73289 do
73290     compound-list
73291 done
```

73292 First, the list of words following **in** shall be expanded to generate a list of items. Then, the  
73293 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no  
73294 items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
73295 in word...
```

73296 shall be equivalent to:

```
73297 in "$@"
```

73298 **Exit Status**

73299 The exit status of a **for** command shall be the exit status of the last command that executes. If  
73300 there are no items, the exit status shall be zero.

73301 2.9.4.3 *Case Conditional Construct*

73302 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of  
73303 several *patterns* (see Section 2.13, on page 2332) that is matched by the string resulting from the  
73304 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote  
73305 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to  
73306 be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol.  
73307 The control operator ')' terminates a list of patterns corresponding to a given action.  
73308 The *compound-list* for each list of patterns, with the possible exception of the last, shall be  
73309 terminated with ";;". The **case** construct terminates with the reserved word **esac** (**case**  
73310 reversed).

73311 The format for the **case** construct is as follows:

```
73312 case word in
73313     [ (]pattern1) compound-list;;
73314     [ [ (]pattern[ | pattern] ... ) compound-list;;] ...
73315     [ [ (]pattern[ | pattern] ... ) compound-list]
73316 esac
```

73317 The ";;" is optional for the last *compound-list*.

73318 In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-*  
73319 *list* shall be subjected to tilde expansion, parameter expansion, command substitution, and  
73320 arithmetic expansion, and the result of these expansions shall be compared against the  
73321 expansion of *word*, according to the rules described in Section 2.13 (on page 2332) (which also  
73322 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall  
73323 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of  
73324 multiple *patterns* that label a *compound-list* statement is unspecified.

73325 **Exit Status**

73326 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be  
73327 the exit status of the last command executed in the *compound-list*.

73328 2.9.4.4 *The if Conditional Construct*

73329 The **if** command shall execute a *compound-list* and use its exit status to determine whether to  
73330 execute another *compound-list*.

73331 The format for the **if** construct is as follows:

```
73332 if compound-list
73333 then
73334     compound-list
73335 [elif compound-list
73336 then
73337     compound-list] ...
73338 [else
73339     compound-list]
73340 fi
```

73341 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be  
 73342 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,  
 73343 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command  
 73344 shall complete. Otherwise, the **else** *compound-list* shall be executed.

#### 73345 **Exit Status**

73346 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that  
 73347 was executed, or zero, if none was executed.

#### 73348 2.9.4.5 *The while Loop*

73349 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 73350 a zero exit status.

73351 The format of the **while** loop is as follows:

```
73352 while compound-list-1
73353 do
73354     compound-list-2
73355 done
```

73356 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command  
 73357 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

#### 73358 **Exit Status**

73359 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or  
 73360 zero if none was executed.

#### 73361 2.9.4.6 *The until Loop*

73362 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 73363 a non-zero exit status.

73364 The format of the **until** loop is as follows:

```
73365 until compound-list-1
73366 do
73367     compound-list-2
73368 done
```

73369 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command  
 73370 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

#### 73371 **Exit Status**

73372 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or  
 73373 zero if none was executed.

73374 **2.9.5 Function Definition Command**

73375 A function is a user-defined name that is used as a simple command to call a compound  
73376 command with new positional parameters. A function is defined with a “function definition  
73377 command”.

73378 The format of a function definition command is as follows:

73379 *fname()* *compound-command*[*io-redirect* ...]

73380 The function is named *fname*; the application shall ensure that it is a name (see XBD Section  
73381 3.230, on page 70). An implementation may allow other characters in a function name as an  
73382 extension. The implementation shall maintain separate name spaces for functions and variables.

73383 The argument *compound-command* represents a compound command, as described in Section  
73384 2.9.4 (on page 2321).

73385 When the function is declared, none of the expansions in Section 2.6 (on page 2305) shall be  
73386 performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as  
73387 normal each time the function is called. Similarly, the optional *io-redirect* redirections and any  
73388 variable assignments within *compound-command* shall be performed during the execution of the  
73389 function itself, not the function definition. See Section 2.8.1 (on page 2315) for the consequences  
73390 of failures of these operations on interactive and non-interactive shells.

73391 When a function is executed, it shall have the syntax-error and variable-assignment properties  
73392 described for special built-in utilities in the enumerated list at the beginning of Section 2.14 (on  
73393 page 2334).

73394 The *compound-command* shall be executed whenever the function name is specified as the name  
73395 of a simple command (see Section 2.9.1.1, on page 2317). The operands to the command  
73396 temporarily shall become the positional parameters during the execution of the *compound-*  
73397 *command*; the special parameter '#' also shall be changed to reflect the number of operands.  
73398 The special parameter 0 shall be unchanged. When the function completes, the values of the  
73399 positional parameters and the special parameter '#' shall be restored to the values they had  
73400 before the function was executed. If the special built-in *return* is executed in the *compound-*  
73401 *command*, the function completes and execution shall resume with the next command after the  
73402 function call.

73403 **Exit Status**

73404 The exit status of a function definition shall be zero if the function was declared successfully;  
73405 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit  
73406 status of the last command executed by the function.

73407 **2.10 Shell Grammar**

73408 The following grammar defines the Shell Command Language. This formal syntax shall take  
73409 precedence over the preceding text syntax description.

73410 **2.10.1 Shell Grammar Lexical Conventions**

73411 The input language to the shell must be first recognized at the character level. The resulting  
73412 tokens shall be classified by their immediate context according to the following rules (applied in  
73413 order). These rules shall be used to determine what a “token” is that is subject to parsing at the  
73414 token level. The rules for token recognition in [Section 2.3](#) (on page 2299) shall apply.

- 73415 1. A <newline> shall be returned as the token identifier **NEWLINE**.
- 73416 2. If the token is an operator, the token identifier for that operator shall result.
- 73417 3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the  
73418 token identifier **IO\_NUMBER** shall be returned.
- 73419 4. Otherwise, the token identifier **TOKEN** results.

73420 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields  
73421 **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the  
73422 context. Some of the productions in the grammar below are annotated with a rule number from  
73423 the following list. When a **TOKEN** is seen where one of those annotated productions could be  
73424 used to reduce the symbol, the applicable rule shall be applied to convert the token identifier  
73425 type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction  
73426 shall then proceed based upon the token identifier type yielded by the rule applied. When more  
73427 than one rule applies, the highest numbered rule shall apply (which in turn may refer to another  
73428 rule). (Note that except in rule 7, the presence of an '=' in the token has no effect.)

73429 The **WORD** tokens shall have the word expansion rules applied to them immediately before the  
73430 associated command is executed, not at the time the command is parsed.

73431 **2.10.2 Shell Grammar Rules**

- 73432 1. [Command Name]

73433 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
73434 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any  
73435 state where only a reserved word could be the next correct token, proceed as above.

73436 **Note:** Because at this point <quotation-mark> characters are retained in the token, quoted  
73437 strings cannot be recognized as reserved words. This rule also implies that reserved  
73438 words are not recognized except in certain positions in the input, such as after a  
73439 <newline> or <semicolon>; the grammar presumes that if the reserved word is  
73440 intended, it is properly delimited by the user, and does not attempt to reflect that  
73441 requirement directly. Also note that line joining is done before tokenization, as described  
73442 in [Section 2.2.1](#) (on page 2298), so escaped <newline> characters are already removed at  
73443 this point.

73444 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or  
73445 applies globally.

- 73446 2. [Redirection to or from filename]

73447 The expansions specified in [Section 2.7](#) (on page 2312) shall occur. As specified there,  
73448 exactly one field can result (or the result is unspecified), and there are additional

- 73449 requirements on pathname expansion.
- 73450 3. [Redirection from here-document]
- 73451 Quote removal shall be applied to the word to determine the delimiter that is used to find  
73452 the end of the here-document that begins after the next <newline>.
- 73453 4. [Case statement termination]
- 73454 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall  
73455 result. Otherwise, the token **WORD** shall be returned.
- 73456 5. [**NAME** in **for**]
- 73457 When the **TOKEN** meets the requirements for a name (see XBD Section 3.230, on page  
73458 70), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be  
73459 returned.
- 73460 6. [Third word of **for** and **case**]
- 73461 a. [**case** only]
- 73462 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall  
73463 result. Otherwise, the token **WORD** shall be returned.
- 73464 b. [**for** only]
- 73465 When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in**  
73466 or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.
- 73467 (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If  
73468 <newline> characters are present at the indicated location, it is the token after them that is  
73469 treated in this fashion.)
- 73470 7. [Assignment preceding command name]
- 73471 a. [When the first word]
- 73472 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b  
73473 shall be applied.
- 73474 b. [Not the first word]
- 73475 If the **TOKEN** contains the <equals-sign> character:
- 73476 If it begins with '=' , the token **WORD** shall be returned.
- 73477 — If all the characters preceding '=' form a valid name (see XBD Section 3.230,  
73478 on page 70), the token **ASSIGNMENT\_WORD** shall be returned. (Quoted  
73479 characters cannot participate in forming a valid name.)
- 73480 — Otherwise, it is unspecified whether it is **ASSIGNMENT\_WORD** or **WORD**  
73481 that is returned.
- 73482 Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 2316).
- 73483 8. [**NAME** in function]
- 73484 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
73485 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token  
73486 identifier **NAME** shall result. Otherwise, rule 7 applies.
- 73487 9. [Body of function]
- 73488 Word expansion and assignment shall never occur, even when required by the rules

73489 above, when this rule is being parsed. Each **TOKEN** that might either be expanded or  
 73490 have assignment applied to it shall instead be returned as a single **WORD** consisting only  
 73491 of characters that are exactly the token described in Section 2.3 (on page 2299).

```

73492 /* -----
73493 The grammar symbols
73494 ----- */
73495 %token WORD
73496 %token ASSIGNMENT_WORD
73497 %token NAME
73498 %token NEWLINE
73499 %token IO_NUMBER

73500 /* The following are the operators mentioned above. */
73501 %token AND_IF OR_IF DSEMI
73502 /* '&&' '||' ';;' */
73503 %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
73504 /* '<<' '>>' '<&' '>&' '<>' '<<-' */
73505 %token CLOBBER
73506 /* '>|' */

73507 /* The following are the reserved words. */
73508 %token If Then Else Elif Fi Do Done
73509 /* 'if' 'then' 'else' 'elif' 'fi' 'do' 'done' */
73510 %token Case Esac While Until For
73511 /* 'case' 'esac' 'while' 'until' 'for' */
73512 /* These are reserved words, not operator tokens, and are
73513 recognized when reserved words are recognized. */
73514 %token Lbrace Rbrace Bang
73515 /* '{' '}' '!' */
73516 %token In
73517 /* 'in' */

73518 /* -----
73519 The Grammar
73520 ----- */
73521 %start complete_command
73522 %%
73523 complete_command : list separator
73524                  | list
73525                  ;
73526 list              : list separator_op and_or
73527                  | and_or
73528                  ;
73529 and_or           : pipeline
73530                  | and_or AND_IF linebreak pipeline
73531                  | and_or OR_IF linebreak pipeline
73532                  ;
73533 pipeline         : pipe_sequence
73534                  | Bang pipe_sequence
73535                  ;

```

```

73536     pipe_sequence      :                               command
73537     | pipe_sequence '|' linebreak command
73538     ;
73539     command              : simple_command
73540     | compound_command
73541     | compound_command redirect_list
73542     | function_definition
73543     ;
73544     compound_command     : brace_group
73545     | subshell
73546     | for_clause
73547     | case_clause
73548     | if_clause
73549     | while_clause
73550     | until_clause
73551     ;
73552     subshell              : '(' compound_list ')'
73553     ;
73554     compound_list        :                               term
73555     | newline_list term
73556     |                               term separator
73557     | newline_list term separator
73558     ;
73559     term                  : term separator and_or
73560     |                               and_or
73561     ;
73562     for_clause            : For name linebreak                               do_group
73563     | For name linebreak in                               sequential_sep do_group
73564     | For name linebreak in wordlist sequential_sep do_group
73565     ;
73566     name                  : NAME                               /* Apply rule 5 */
73567     ;
73568     in                    : In                               /* Apply rule 6 */
73569     ;
73570     wordlist              : wordlist WORD
73571     |                               WORD
73572     ;
73573     case_clause           : Case WORD linebreak in linebreak case_list      Esac
73574     | Case WORD linebreak in linebreak case_list_ns Esac
73575     | Case WORD linebreak in linebreak                               Esac
73576     ;
73577     case_list_ns          : case_list case_item_ns
73578     |                               case_item_ns
73579     ;
73580     case_list              : case_list case_item
73581     |                               case_item
73582     ;
73583     case_item_ns          : pattern ')' linebreak
73584     | pattern ')' compound_list linebreak
73585     | '(' pattern ')' linebreak
73586     | '(' pattern ')' compound_list linebreak
73587     ;
73588     case_item              : pattern ')' linebreak      DSEMI linebreak

```

```

73589         |      pattern ')' compound_list DSEMI linebreak
73590         | ' (' pattern ')' linebreak      DSEMI linebreak
73591         | ' (' pattern ')' compound_list DSEMI linebreak
73592         ;
73593 pattern      :      WORD      /* Apply rule 4 */
73594         | pattern '|' WORD      /* Do not apply rule 4 */
73595         ;
73596 if_clause    : If compound_list Then compound_list else_part Fi
73597         | If compound_list Then compound_list      Fi
73598         ;
73599 else_part    : Elif compound_list Then else_part
73600         | Else compound_list
73601         ;
73602 while_clause : While compound_list do_group
73603         ;
73604 until_clause : Until compound_list do_group
73605         ;
73606 function_definition : fname '(' ')' linebreak function_body
73607         ;
73608 function_body  : compound_command      /* Apply rule 9 */
73609         | compound_command redirect_list /* Apply rule 9 */
73610         ;
73611 fname          : NAME      /* Apply rule 8 */
73612         ;
73613 brace_group    : Lbrace compound_list Rbrace
73614         ;
73615 do_group       : Do compound_list Done      /* Apply rule 6 */
73616         ;
73617 simple_command : cmd_prefix cmd_word cmd_suffix
73618         | cmd_prefix cmd_word
73619         | cmd_prefix
73620         | cmd_name cmd_suffix
73621         | cmd_name
73622         ;
73623 cmd_name       : WORD      /* Apply rule 7a */
73624         ;
73625 cmd_word       : WORD      /* Apply rule 7b */
73626         ;
73627 cmd_prefix     :      io_redirect
73628         | cmd_prefix io_redirect
73629         |      ASSIGNMENT_WORD
73630         | cmd_prefix ASSIGNMENT_WORD
73631         ;
73632 cmd_suffix     :      io_redirect
73633         | cmd_suffix io_redirect
73634         |      WORD
73635         | cmd_suffix WORD
73636         ;
73637 redirect_list :      io_redirect
73638         | redirect_list io_redirect
73639         ;
73640 io_redirect    :      io_file
73641         | IO_NUMBER io_file

```

```

73642         |             io_here
73643         | IO_NUMBER io_here
73644         ;
73645     io_file      : '<'         filename
73646         | LESSAND   filename
73647         | '>'         filename
73648         | GREATAND  filename
73649         | DGREAT   filename
73650         | LESSGREAT filename
73651         | CLOBBER  filename
73652         ;
73653     filename     : WORD                               /* Apply rule 2 */
73654         ;
73655     io_here      : DLESS      here_end
73656         | DLESSDASH here_end
73657         ;
73658     here_end     : WORD                               /* Apply rule 3 */
73659         ;
73660     newline_list :             NEWLINE
73661         | newline_list NEWLINE
73662         ;
73663     linebreak    : newline_list
73664         | /* empty */
73665         ;
73666     separator_op : '&'
73667         | ';'
73668         ;
73669     separator    : separator_op linebreak
73670         | newline_list
73671         ;
73672     sequential_sep : ';' linebreak
73673         | newline_list
73674         ;

```

## 73675 2.11 Signals and Error Handling

73676 When a command is in an asynchronous list, it shall inherit from the shell a signal action of  
73677 ignored (SIG\_IGN) for the SIGQUIT and SIGINT signals, and may inherit a signal mask in  
73678 which SIGQUIT and SIGINT are blocked. Otherwise, the signal actions and signal mask  
73679 inherited by the command shall be the same as those inherited by the shell from its parent  
73680 unless a signal action is modified by the *trap* special built-in (see *trap*)

73681 When a signal for which a trap has been set is received while the shell is waiting for the  
73682 completion of a utility executing a foreground command, the trap associated with that signal  
73683 shall not be executed until after the foreground command has completed. When the shell is  
73684 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a  
73685 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit  
73686 status >128, immediately after which the trap associated with that signal shall be taken.

73687 If multiple signals are pending for the shell for which there are associated trap actions, the order  
73688 of execution of trap actions is unspecified.

73689 **2.12 Shell Execution Environment**

73690 A shell execution environment consists of the following:

- 73691 • Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- 73692 • Working directory as set by *cd*
- 73693 • File creation mask set by *umask*
- 73694 • Current traps set by *trap*
- 73695 • Shell parameters that are set by variable assignment (see the *set* special built-in) or from
- 73696 the System Interfaces volume of POSIX.1-2008 environment inherited by the shell when it
- 73697 begins (see the *export* special built-in)
- 73698 • Shell functions; see Section 2.9.5 (on page 2324)
- 73699 • Options turned on at invocation or by *set*
- 73700 • Process IDs of the last commands in asynchronous lists known to this shell environment;
- 73701 see Section 2.9.3.1 (on page 2319)
- 73702 • Shell aliases; see Section 2.3.1 (on page 2300)

73703 Utilities other than the special built-ins (see Section 2.14, on page 2334) shall be invoked in a  
 73704 separate environment that consists of the following. The initial value of these objects shall be the  
 73705 same as that for the parent shell, except as noted below.

- 73706 • Open files inherited on invocation of the shell, open files controlled by the *exec* special  
 73707 built-in plus any modifications, and additions specified by any redirections to the utility
- 73708 • Current working directory
- 73709 • File creation mask
- 73710 • If the utility is a shell script, traps caught by the shell shall be set to the default values and  
 73711 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell  
 73712 script, the trap actions (default or ignore) shall be mapped into the appropriate signal  
 73713 handling actions for the utility
- 73714 • Variables with the *export* attribute, along with those explicitly exported for the duration of  
 73715 the command, shall be passed to the utility environment variables

73716 The environment of the shell process shall not be changed by the utility unless explicitly  
 73717 specified by the utility description (for example, *cd* and *umask*).

73718 A subshell environment shall be created as a duplicate of the shell environment, except that  
 73719 signal traps set by that shell environment shall be set to the default values. Changes made to the  
 73720 subshell environment shall not affect the shell environment. Command substitution, commands  
 73721 that are grouped with parentheses, and asynchronous lists shall be executed in a subshell  
 73722 environment. Additionally, each command of a multi-command pipeline is in a subshell  
 73723 environment; as an extension, however, any or all commands in a pipeline may be executed in  
 73724 the current environment. All other commands shall be executed in the current shell  
 73725 environment.

## 73726 2.13 Pattern Matching Notation

73727 The pattern matching notation described in this section is used to specify patterns for matching  
 73728 strings in the shell. Historically, pattern matching notation is related to, but slightly different  
 73729 from, the regular expression notation described in XBD [Chapter 9](#) (on page 181). For this reason,  
 73730 the description of the rules for this pattern matching notation are based on the description of  
 73731 regular expression notation, modified to account for the differences.

### 73732 2.13.1 Patterns Matching a Single Character

73733 The following patterns matching a single character shall match a single character: ordinary  
 73734 characters, special pattern characters, and pattern bracket expressions. The pattern bracket  
 73735 expression also shall match a single collating element. A <backslash> character shall escape the  
 73736 following character. The escaping <backslash> shall be discarded.

73737 An ordinary character is a pattern that shall match itself. It can be any character in the supported  
 73738 character set except for NUL, those special shell characters in [Section 2.2](#) (on page 2298) that  
 73739 require quoting, and the following three special pattern characters. Matching shall be based on  
 73740 the bit pattern used for encoding the character, not on the graphic representation of the  
 73741 character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall  
 73742 match the character itself. The shell special characters always require quoting.

73743 When unquoted and outside a bracket expression, the following three characters shall have  
 73744 special meaning in the specification of patterns:

- 73745 ? A <question-mark> is a pattern that shall match any character.
- 73746 \* An <asterisk> is a pattern that shall match multiple characters, as described in [Section](#)  
 73747 [2.13.2](#).
- 73748 [ If an open bracket introduces a bracket expression as in XBD [Section 9.3.5](#) (on page 184),  
 73749 except that the <exclamation-mark> character ('!') shall replace the <circumflex>  
 73750 character ('^') in its role in a non-matching list in the regular expression notation, it shall  
 73751 introduce a pattern bracket expression. A bracket expression starting with an unquoted  
 73752 <circumflex> character produces unspecified results. Otherwise, '[' shall match the  
 73753 character itself.

73754 When pattern matching is used where shell quote removal is not performed (such as in the  
 73755 argument to the *find -name* primary when *find* is being called using one of the *exec* functions as  
 73756 defined in the System Interfaces volume of POSIX.1-2008, or in the *pattern* argument to the  
 73757 *fnmatch()* function), special characters can be escaped to remove their special meaning by  
 73758 preceding them with a <backslash> character. This escaping <backslash> is discarded. The  
 73759 sequence "\\\" represents one literal <backslash>. All of the requirements and effects of quoting  
 73760 on ordinary, shell special, and special pattern characters shall apply to escaping in this context.

### 73761 2.13.2 Patterns Matching Multiple Characters

73762 The following rules are used to construct patterns matching multiple characters from patterns  
 73763 matching a single character:

- 73764 1. The <asterisk> ('\*') is a pattern that shall match any string, including the null string.
- 73765 2. The concatenation of patterns matching a single character is a valid pattern that shall  
 73766 match the concatenation of the single characters or collating elements matched by each of  
 73767 the concatenated patterns.

- 73768 3. The concatenation of one or more patterns matching a single character with one or more  
 73769 <asterisk> characters is a valid pattern. In such patterns, each <asterisk> shall match a  
 73770 string of zero or more characters, matching the greatest possible number of characters  
 73771 that still allows the remainder of the pattern to match the string.

### 73772 2.13.3 Patterns Used for Filename Expansion

73773 The rules described so far in Section 2.13.1 (on page 2332) and Section 2.13.2 (on page 2332) are  
 73774 qualified by the following rules that apply when pattern matching notation is used for filename  
 73775 expansion:

- 73776 1. The <slash> character in a pathname shall be explicitly matched by using one or more  
 73777 <slash> characters in the pattern; it shall neither be matched by the <asterisk> or  
 73778 <question-mark> special characters nor by a bracket expression. <slash> characters in the  
 73779 pattern shall be identified before bracket expressions; thus, a <slash> cannot be included  
 73780 in a pattern bracket expression used for filename expansion. If a <slash> character is  
 73781 found following an unescaped <left-square-bracket> character before a corresponding  
 73782 <right-square-bracket> is found, the open bracket shall be treated as an ordinary  
 73783 character. For example, the pattern "a[b/c]d" does not match such pathnames as **abd**  
 73784 or **a/d**. It only matches a pathname of literally **a[b/c]d**.
- 73785 2. If a filename begins with a <period> ( ' . ' ), the <period> shall be explicitly matched by  
 73786 using a <period> as the first character of the pattern or immediately following a <slash>  
 73787 character. The leading <period> shall not be matched by:
- 73788 • The <asterisk> or <question-mark> special characters
  - 73789 • A bracket expression containing a non-matching list, such as "[!a]", a range  
 73790 expression, such as "[%=0]", or a character class expression, such as  
 73791 "[[:punct:]]"

73792 It is unspecified whether an explicit <period> in a bracket expression matching list, such  
 73793 as "[.abc]", can match a leading <period> in a filename.

- 73794 3. Specified patterns shall be matched against existing filenames and pathnames, as  
 73795 appropriate. Each component that contains a pattern character shall require read  
 73796 permission in the directory containing that component. Any component, except the last,  
 73797 that does not contain a pattern character shall require search permission. For example,  
 73798 given the pattern:

73799 /foo/bar/x\*/bam

73800 search permission is needed for directories / and **foo**, search and read permissions are  
 73801 needed for directory **bar**, and search permission is needed for each **x\*** directory. If the  
 73802 pattern matches any existing filenames or pathnames, the pattern shall be replaced with  
 73803 those filenames and pathnames, sorted according to the collating sequence in effect in the  
 73804 current locale.

73805 If the pattern contains an open bracket ( ' [ ' ) that does not introduce a bracket expression  
 73806 as in XBD Section 9.3.5 (on page 184), it is unspecified whether other unquoted pattern  
 73807 matching characters within the same slash-delimited component of the pattern retain  
 73808 their special meanings or are treated as ordinary characters. For example, the pattern  
 73809 "a\*[/b\*" may match all filenames beginning with ' b ' in the directory "a\*[" or it may  
 73810 match all filenames beginning with ' b ' in all directories with names beginning with ' a '  
 73811 and ending with ' [ ' .

73812 If the pattern does not match any existing filenames or pathnames, the pattern string shall

73813 be left unchanged.

## 73814 2.14 Special Built-In Utilities

73815 The following “special built-in” utilities shall be supported in the shell command language. The  
73816 output of each command, if any, shall be written to standard output, subject to the normal  
73817 redirection and piping possible with all commands.

73818 The term “built-in” implies that the shell can execute the utility directly and does not need to  
73819 search for it. An implementation may choose to make any utility a built-in; however, the special  
73820 built-in utilities described here differ from regular built-in utilities in two respects:

- 73821 1. A syntax error in a special built-in utility may cause a shell executing that utility to abort,  
73822 while a syntax error in a regular built-in utility shall not cause a shell executing that  
73823 utility to abort. (See [Section 2.8.1](#) (on page 2315) for the consequences of errors on  
73824 interactive and non-interactive shells.) If a special built-in utility encountering a syntax  
73825 error does not abort the shell, its exit value shall be non-zero.
- 73826 2. Variable assignments specified with special built-in utilities remain in effect after the  
73827 built-in completes; this shall not be the case with a regular built-in or other utility.

73828 The special built-in utilities in this section need not be provided in a manner accessible via the  
73829 *exec* family of functions defined in the System Interfaces volume of POSIX.1-2008.

73830 Some of the special built-ins are described as conforming to XBD [Section 12.2](#) (on page 215). For  
73831 those that are not, the requirement in [Section 1.4](#) (on page 2288) that “--” be recognized as a  
73832 first argument to be discarded does not apply and a conforming application shall not use that  
73833 argument.

73834 **NAME**

73835 break — exit from for, while, or until loop

73836 **SYNOPSIS**73837 break [*n*]73838 **DESCRIPTION**

73839 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from  
 73840 the *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater  
 73841 than or equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of  
 73842 enclosing loops, the outermost enclosing loop shall be exited. Execution shall continue with the  
 73843 command immediately following the loop.

73844 **OPTIONS**

73845 None.

73846 **OPERANDS**

73847 See the DESCRIPTION.

73848 **STDIN**

73849 Not used.

73850 **INPUT FILES**

73851 None.

73852 **ENVIRONMENT VARIABLES**

73853 None.

73854 **ASYNCHRONOUS EVENTS**

73855 Default.

73856 **STDOUT**

73857 Not used.

73858 **STDERR**

73859 The standard error shall be used only for diagnostic messages.

73860 **OUTPUT FILES**

73861 None.

73862 **EXTENDED DESCRIPTION**

73863 None.

73864 **EXIT STATUS**

73865 0 Successful completion.

73866 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.73867 **CONSEQUENCES OF ERRORS**

73868 Default.

**break**73869 **APPLICATION USAGE**

73870 None.

73871 **EXAMPLES**

```

73872     for i in *
73873     do
73874         if test -d "$i"
73875         then break
73876         fi
73877     done

```

73878 **RATIONALE**

73879 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer  
 73880 to a label associated with the appropriate loop as a preferable alternative to the *n* method.  
 73881 However, this volume of POSIX.1-2008 does reserve the name space of command names ending  
 73882 with a <colon>. It is anticipated that a future implementation could take advantage of this and  
 73883 provide something like:

```

73884     outofloop: for i in a b c d e
73885     do
73886         for j in 0 1 2 3 4 5 6 7 8 9
73887         do
73888             if test -r "${i}${j}"
73889             then break outofloop
73890             fi
73891         done
73892     done

```

73893 and that this might be standardized after implementation experience is achieved.

73894 **FUTURE DIRECTIONS**

73895 None.

73896 **SEE ALSO**73897 [Section 2.14](#) (on page 2334)73898 **CHANGE HISTORY**73899 **Issue 6**

73900 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
 73901 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
 73902 behavior is intended.

73903 **NAME**

73904 colon — null utility

73905 **SYNOPSIS**73906 : [*argument*...]73907 **DESCRIPTION**73908 This utility shall only expand command *arguments*. It is used when a command is needed, as in  
73909 the **then** condition of an **if** command, but nothing is to be done by the command.73910 **OPTIONS**

73911 None.

73912 **OPERANDS**

73913 See the DESCRIPTION.

73914 **STDIN**

73915 Not used.

73916 **INPUT FILES**

73917 None.

73918 **ENVIRONMENT VARIABLES**

73919 None.

73920 **ASYNCHRONOUS EVENTS**

73921 Default.

73922 **STDOUT**

73923 Not used.

73924 **STDERR**

73925 The standard error shall be used only for diagnostic messages.

73926 **OUTPUT FILES**

73927 None.

73928 **EXTENDED DESCRIPTION**

73929 None.

73930 **EXIT STATUS**

73931 Zero.

73932 **CONSEQUENCES OF ERRORS**

73933 Default.

73934 **APPLICATION USAGE**

73935 None.

73936 **EXAMPLES**

```

73937 : ${X=abc}
73938 if     false
73939 then  :
73940 else  echo $X
73941 fi
73942 abc

```

73943 As with any of the special built-ins, the null utility can also have variable assignments and  
73944 redirections associated with it, such as:73945 `x=y : > z`

**colon***Shell Command Language*

73946 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates  
73947 or truncates file *z*.

**RATIONALE**

73948 None.  
73949

**FUTURE DIRECTIONS**

73950 None.  
73951

**SEE ALSO**

73952 [Section 2.14](#) (on page 2334)  
73953

**CHANGE HISTORY****Issue 6**

73954 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
73955 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
73956 behavior is intended.  
73957  
73958

**Issue 7**

73959 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.  
73960

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

73961 **NAME**

73962 continue — continue for, while, or until loop

73963 **SYNOPSIS**73964 continue [*n*]73965 **DESCRIPTION**

73966 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to  
 73967 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a  
 73968 **while** or **until** loop or performing the next assignment of a **for** loop, and re-executing the loop if  
 73969 appropriate.

73970 The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to  
 73971 *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be  
 73972 used.

73973 **OPTIONS**

73974 None.

73975 **OPERANDS**

73976 See the DESCRIPTION.

73977 **STDIN**

73978 Not used.

73979 **INPUT FILES**

73980 None.

73981 **ENVIRONMENT VARIABLES**

73982 None.

73983 **ASYNCHRONOUS EVENTS**

73984 Default.

73985 **STDOUT**

73986 Not used.

73987 **STDERR**

73988 The standard error shall be used only for diagnostic messages.

73989 **OUTPUT FILES**

73990 None.

73991 **EXTENDED DESCRIPTION**

73992 None.

73993 **EXIT STATUS**

73994 0 Successful completion.

73995 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.73996 **CONSEQUENCES OF ERRORS**

73997 Default.

**continue**

Shell Command Language

73998 **APPLICATION USAGE**

73999 None.

74000 **EXAMPLES**

```
74001     for i in *
74002     do
74003         if test -d "$i"
74004         then continue
74005         fi
74006         printf '%s' is not a directory.\n' "$i"
74007     done
```

74008 **RATIONALE**

74009 None.

74010 **FUTURE DIRECTIONS**

74011 None.

74012 **SEE ALSO**74013 [Section 2.14](#) (on page 2334)74014 **CHANGE HISTORY**74015 **Issue 6**

74016 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74017 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74018 behavior is intended.

74019 **Issue 7**74020 The example is changed to use the *printf* utility rather than *echo*.

74021 **NAME**

74022 dot — execute commands in the current environment

74023 **SYNOPSIS**74024 . *file*74025 **DESCRIPTION**74026 The shell shall execute commands from the *file* in the current environment.

74027 If *file* does not contain a <slash>, the shell shall use the search path specified by *PATH* to find the  
 74028 directory containing *file*. Unlike normal command search, however, the file searched for by the  
 74029 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;  
 74030 an interactive shell shall write a diagnostic message to standard error, but this condition shall  
 74031 not be considered a syntax error.

74032 **OPTIONS**

74033 None.

74034 **OPERANDS**

74035 See the DESCRIPTION.

74036 **STDIN**

74037 Not used.

74038 **INPUT FILES**

74039 See the DESCRIPTION.

74040 **ENVIRONMENT VARIABLES**

74041 See the DESCRIPTION.

74042 **ASYNCHRONOUS EVENTS**

74043 Default.

74044 **STDOUT**

74045 Not used.

74046 **STDERR**

74047 The standard error shall be used only for diagnostic messages.

74048 **OUTPUT FILES**

74049 None.

74050 **EXTENDED DESCRIPTION**

74051 None.

74052 **EXIT STATUS**

74053 Returns the value of the last command executed, or a zero exit status if no command is executed.

74054 **CONSEQUENCES OF ERRORS**

74055 Default.

**dot**74056 **APPLICATION USAGE**

74057 None.

74058 **EXAMPLES**74059 `cat foobar`74060 `foo=hello bar=world`74061 `./foobar`74062 `echo $foo $bar`74063 `hello world`74064 **RATIONALE**

74065 Some older implementations searched the current directory for the *file*, even if the value of *PATH*  
74066 disallowed it. This behavior was omitted from this volume of POSIX.1-2008 due to concerns  
74067 about introducing the susceptibility to trojan horses that the user might be trying to avoid by  
74068 leaving **dot** out of *PATH*.

74069 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.  
74070 This is a valid extension that allows a *dot* script to behave identically to a function.

74071 **FUTURE DIRECTIONS**

74072 None.

74073 **SEE ALSO**74074 [Section 2.14](#) (on page 2334)74075 **CHANGE HISTORY**74076 **Issue 6**

74077 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74078 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74079 behavior is intended.

74080 **Issue 7**

74081 SD5-XCU-ERN-164 is applied.

74082 **NAME**

74083 eval — construct command by concatenating arguments

74084 **SYNOPSIS**74085 eval [*argument*...]74086 **DESCRIPTION**74087 The *eval* utility shall construct a command by concatenating *arguments* together, separating each  
74088 with a <space> character. The constructed command shall be read and executed by the shell.74089 **OPTIONS**

74090 None.

74091 **OPERANDS**

74092 See the DESCRIPTION.

74093 **STDIN**

74094 Not used.

74095 **INPUT FILES**

74096 None.

74097 **ENVIRONMENT VARIABLES**

74098 None.

74099 **ASYNCHRONOUS EVENTS**

74100 Default.

74101 **STDOUT**

74102 Not used.

74103 **STDERR**

74104 The standard error shall be used only for diagnostic messages.

74105 **OUTPUT FILES**

74106 None.

74107 **EXTENDED DESCRIPTION**

74108 None.

74109 **EXIT STATUS**74110 If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it  
74111 shall return the exit status of the command defined by the string of concatenated *arguments*  
74112 separated by <space> characters.74113 **CONSEQUENCES OF ERRORS**

74114 Default.

74115 **APPLICATION USAGE**

74116 None.

74117 **EXAMPLES**

74118 foo=10 x=foo

74119 y=' \$' \$x

74120 echo \$y

74121 **\$foo**

74122 eval y=' \$' \$x

74123 echo \$y

74124 **10**

**eval***Shell Command Language*74125 **RATIONALE**

74126 None.

74127 **FUTURE DIRECTIONS**

74128 None.

74129 **SEE ALSO**74130 [Section 2.14](#) (on page 2334)74131 **CHANGE HISTORY**74132 **Issue 6**

74133 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74134 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74135 behavior is intended.

74136 **Issue 7**

74137 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

74138 **NAME**74139 `exec` — execute commands and open, close, or copy file descriptors74140 **SYNOPSIS**74141 `exec [command [argument...]]`74142 **DESCRIPTION**74143 The `exec` utility shall open, close, and/or copy file descriptors as specified by any redirections as  
74144 part of the command.74145 If `exec` is specified without `command` or `arguments`, and any file descriptors with numbers greater  
74146 than 2 are opened with associated redirection statements, it is unspecified whether those file  
74147 descriptors remain open when the shell invokes another utility. Scripts concerned that child  
74148 shells could misuse open file descriptors can always close them explicitly, as shown in one of the  
74149 following examples.74150 If `exec` is specified with `command`, it shall replace the shell with `command` without creating a new  
74151 process. If `arguments` are specified, they shall be arguments to `command`. Redirection affects the  
74152 current shell execution environment.74153 **OPTIONS**

74154 None.

74155 **OPERANDS**

74156 See the DESCRIPTION.

74157 **STDIN**

74158 Not used.

74159 **INPUT FILES**

74160 None.

74161 **ENVIRONMENT VARIABLES**

74162 None.

74163 **ASYNCHRONOUS EVENTS**

74164 Default.

74165 **STDOUT**

74166 Not used.

74167 **STDERR**

74168 The standard error shall be used only for diagnostic messages.

74169 **OUTPUT FILES**

74170 None.

74171 **EXTENDED DESCRIPTION**

74172 None.

74173 **EXIT STATUS**74174 If `command` is specified, `exec` shall not return to the shell; rather, the exit status of the process shall  
74175 be the exit status of the program implementing `command`, which overlaid the shell. If `command` is  
74176 not found, the exit status shall be 127. If `command` is found, but it is not an executable utility, the  
74177 exit status shall be 126. If a redirection error occurs (see [Section 2.8.1](#), on page 2315), the shell  
74178 shall exit with a value in the range 1–125. Otherwise, `exec` shall return a zero exit status.

74179 **CONSEQUENCES OF ERRORS**

74180 Default.

74181 **APPLICATION USAGE**

74182 None.

74183 **EXAMPLES**74184 Open *readfile* as file descriptor 3 for reading:74185 `exec 3< readfile`74186 Open *writefile* as file descriptor 4 for writing:74187 `exec 4> writefile`

74188 Make file descriptor 5 a copy of file descriptor 0:

74189 `exec 5<&0`

74190 Close file descriptor 3:

74191 `exec 3<&-`74192 Cat the file **maggie** by replacing the current shell with the *cat* utility:74193 `exec cat maggie`74194 **RATIONALE**

74195 Most historical implementations were not conformant in that:

74196 `foo=bar exec cmd`74197 did not pass **foo** to **cmd**.74198 **FUTURE DIRECTIONS**

74199 None.

74200 **SEE ALSO**74201 [Section 2.14](#) (on page 2334)74202 **CHANGE HISTORY**74203 **Issue 6**74204 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74205 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74206 behavior is intended.74207 **Issue 7**

74208 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74209 **NAME**

74210 exit — cause the shell to exit

74211 **SYNOPSIS**74212 exit [*n*]74213 **DESCRIPTION**

74214 The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal  
 74215 integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is  
 74216 undefined.

74217 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is  
 74218 invoked in that *trap* itself, in which case the shell shall exit immediately.

74219 **OPTIONS**

74220 None.

74221 **OPERANDS**

74222 See the DESCRIPTION.

74223 **STDIN**

74224 Not used.

74225 **INPUT FILES**

74226 None.

74227 **ENVIRONMENT VARIABLES**

74228 None.

74229 **ASYNCHRONOUS EVENTS**

74230 Default.

74231 **STDOUT**

74232 Not used.

74233 **STDERR**

74234 The standard error shall be used only for diagnostic messages.

74235 **OUTPUT FILES**

74236 None.

74237 **EXTENDED DESCRIPTION**

74238 None.

74239 **EXIT STATUS**

74240 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last  
 74241 command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,  
 74242 the last command is considered to be the command that executed immediately preceding the  
 74243 *trap* action.

74244 **CONSEQUENCES OF ERRORS**

74245 Default.

**exit***Shell Command Language*74246 **APPLICATION USAGE**

74247 None.

74248 **EXAMPLES**74249 Exit with a *true* value:74250 `exit 0`74251 Exit with a *false* value:74252 `exit 1`74253 **RATIONALE**74254 As explained in other sections, certain exit status values have been reserved for special uses and  
74255 should be used by applications only for those purposes:

74256 126 A file to be executed was found, but it was not an executable utility.

74257 127 A utility to be executed was not found.

74258 &gt;128 A command was interrupted by a signal.

74259 **FUTURE DIRECTIONS**

74260 None.

74261 **SEE ALSO**74262 [Section 2.14](#) (on page 2334)74263 **CHANGE HISTORY**74264 **Issue 6**74265 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74266 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74267 behavior is intended.

74268 **NAME**

74269 export — set the export attribute for variables

74270 **SYNOPSIS**

74271 export name [=word] . . .

74272 export -p

74273 **DESCRIPTION**74274 The shell shall give the *export* attribute to the variables corresponding to the specified *names*,  
74275 which shall cause them to be in the environment of subsequently executed commands. If the  
74276 name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.74277 The *export* special built-in shall support XBD Section 12.2 (on page 215).74278 When **-p** is specified, *export* shall write to the standard output the names and values of all  
74279 exported variables, in the following format:

74280 "export %s=%s\n", &lt;name&gt;, &lt;value&gt;

74281 if *name* is set, and:

74282 "export %s\n", &lt;name&gt;

74283 if *name* is unset.74284 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
74285 reinput to the shell as commands that achieve the same exporting results, except:

- 74286 1. Read-only variables with values cannot be reset.
- 
- 74287 2. Variables that were unset at the time they were output need not be reset to the unset state
- 
- 74288 if a value is assigned to the variable between the time the state was saved and the time at
- 
- 74289 which the saved output is reinput to the shell.

74290 When no arguments are given, the results are unspecified.

74291 **OPTIONS**

74292 See the DESCRIPTION.

74293 **OPERANDS**

74294 See the DESCRIPTION.

74295 **STDIN**

74296 Not used.

74297 **INPUT FILES**

74298 None.

74299 **ENVIRONMENT VARIABLES**

74300 None.

74301 **ASYNCHRONOUS EVENTS**

74302 Default.

74303 **STDOUT**

74304 See the DESCRIPTION.

74305 **STDERR**

74306 The standard error shall be used only for diagnostic messages.

**export**74307 **OUTPUT FILES**

74308 None.

74309 **EXTENDED DESCRIPTION**

74310 None.

74311 **EXIT STATUS**

74312 Zero.

74313 **CONSEQUENCES OF ERRORS**

74314 Default.

74315 **APPLICATION USAGE**

74316 None.

74317 **EXAMPLES**74318 Export *PWD* and *HOME* variables:74319 

```
export PWD HOME
```

74320 Set and export the *PATH* variable:74321 

```
export PATH=/local/bin:$PATH
```

74322 Save and restore all exported variables:

```
74323 export -p > temp-file
74324 unset a lot of variables
74325 ... processing
74326 . temp-file
```

74327 **RATIONALE**

74328 Some historical shells use the no-argument case as the functional equivalent of what is required  
 74329 here with **-p**. This feature was left unspecified because it is not historical practice in all shells,  
 74330 and some scripts may rely on the now-unspecified results on their implementations. Attempts to  
 74331 specify the **-p** output as the default case were unsuccessful in achieving consensus. The **-p**  
 74332 option was added to allow portable access to the values that can be saved and then later restored  
 74333 using; for example, a *dot* script.

74334 **FUTURE DIRECTIONS**

74335 None.

74336 **SEE ALSO**74337 [Section 2.14](#) (on page 2334)74338 [XBD Section 12.2](#) (on page 215)74339 **CHANGE HISTORY**74340 **Issue 6**

74341 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

74342 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
 74343 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
 74344 behavior is intended.

74345 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to  
 74346 the end of the first paragraph of the DESCRIPTION: "If the name of a variable is followed by  
 74347 =*word*, then the value of that variable shall be set to *word*". The reason for this change is that the  
 74348 SYNOPSIS for *export* includes:

74349        `export name [=word] . . .`

74350        but the meaning of the optional “=word” is never explained in the text.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**readonly**

Shell Command Language

74351 **NAME**

74352           readonly — set the readonly attribute for variables

74353 **SYNOPSIS**

74354           readonly name [=word] . . .

74355           readonly -p

74356 **DESCRIPTION**

74357           The variables whose *names* are specified shall be given the *readonly* attribute. The values of  
 74358           variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those  
 74359           variables be unset by the *unset* utility. If the name of a variable is followed by =*word*, then the  
 74360           value of that variable shall be set to *word*.

74361           The *readonly* special built-in shall support XBD [Section 12.2](#) (on page 215).

74362           When **-p** is specified, *readonly* writes to the standard output the names and values of all read-  
 74363           only variables, in the following format:

74364           "readonly %s=%s\n", &lt;name&gt;, &lt;value&gt;

74365           if *name* is set, and

74366           "readonly %s\n", &lt;name&gt;

74367           if *name* is unset.

74368           The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 74369           reinput to the shell as commands that achieve the same value and *readonly* attribute-setting  
 74370           results in a shell execution environment in which:

- 74371           1. Variables with values at the time they were output do not have the *readonly* attribute set.
- 74372           2. Variables that were unset at the time they were output do not have a value at the time at  
 74373           which the saved output is reinput to the shell.

74374           When no arguments are given, the results are unspecified.

74375 **OPTIONS**

74376           See the DESCRIPTION.

74377 **OPERANDS**

74378           See the DESCRIPTION.

74379 **STDIN**

74380           Not used.

74381 **INPUT FILES**

74382           None.

74383 **ENVIRONMENT VARIABLES**

74384           None.

74385 **ASYNCHRONOUS EVENTS**

74386           Default.

74387 **STDOUT**

74388           See the DESCRIPTION.

74389 **STDERR**

74390 The standard error shall be used only for diagnostic messages.

74391 **OUTPUT FILES**

74392 None.

74393 **EXTENDED DESCRIPTION**

74394 None.

74395 **EXIT STATUS**

74396 Zero.

74397 **CONSEQUENCES OF ERRORS**

74398 Default.

74399 **APPLICATION USAGE**

74400 None.

74401 **EXAMPLES**74402 `readonly HOME PWD`74403 **RATIONALE**74404 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of  
74405 POSIX.1-2008 allows this behavior, but does not require it.74406 The `-p` option allows portable access to the values that can be saved and then later restored  
74407 using, for example, a *dot* script. Also see the RATIONALE for *export* (on page 2349) for a  
74408 description of the no-argument and `-p` output cases and a related example.74409 Read-only functions were considered, but they were omitted as not being historical practice or  
74410 particularly useful. Furthermore, functions must not be read-only across invocations to preclude  
74411 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-  
74412 known utility with the intent of subverting the real intent of the user) of administrative or  
74413 security-relevant (or security-conscious) shell scripts.74414 **FUTURE DIRECTIONS**

74415 None.

74416 **SEE ALSO**74417 [Section 2.14](#) (on page 2334)74418 XBD [Section 12.2](#) (on page 215)74419 **CHANGE HISTORY**74420 **Issue 6**

74421 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

74422 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74423 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74424 behavior is intended.74425 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to  
74426 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by  
74427 `=word`, then the value of that variable shall be set to `word`.”. The reason for this change is that the  
74428 SYNOPSIS for *readonly* includes:

## readonly

*Shell Command Language*

74429        `readonly name [=word] . . .`

74430        but the meaning of the optional “=word” is never explained in the text.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

74431 **NAME**

74432 return — return from a function

74433 **SYNOPSIS**74434 return [*n*]74435 **DESCRIPTION**74436 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the  
74437 shell is not currently executing a function or *dot* script, the results are unspecified.74438 **OPTIONS**

74439 None.

74440 **OPERANDS**

74441 See the DESCRIPTION.

74442 **STDIN**

74443 Not used.

74444 **INPUT FILES**

74445 None.

74446 **ENVIRONMENT VARIABLES**

74447 None.

74448 **ASYNCHRONOUS EVENTS**

74449 Default.

74450 **STDOUT**

74451 Not used.

74452 **STDERR**

74453 The standard error shall be used only for diagnostic messages.

74454 **OUTPUT FILES**

74455 None.

74456 **EXTENDED DESCRIPTION**

74457 None.

74458 **EXIT STATUS**74459 The value of the special parameter '?' shall be set to *n*, an unsigned decimal integer, or to the  
74460 exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,  
74461 the results are undefined. When *return* is executed in a *trap* action, the last command is  
74462 considered to be the command that executed immediately preceding the *trap* action.74463 **CONSEQUENCES OF ERRORS**

74464 Default.

74465 **APPLICATION USAGE**

74466 None.

74467 **EXAMPLES**

74468 None.

74469 **RATIONALE**74470 The behavior of *return* when not in a function or *dot* script differs between the System V shell  
74471 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is  
74472 the same as *exit*.

74473 The results of returning a number greater than 255 are undefined because of differing practices

**return***Shell Command Language*

74474 in the various historical implementations. Some shells AND out all but the low-order 8 bits;  
74475 others allow larger values, but not of unlimited size.

74476 See the discussion of appropriate exit status values under *exit* (on page 2347).

**74477 FUTURE DIRECTIONS**

74478 None.

**74479 SEE ALSO**

74480 [Section 2.14](#) (on page 2334)

**74481 CHANGE HISTORY****74482 Issue 6**

74483 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74484 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74485 behavior is intended.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

74486 **NAME**

74487 set — set or unset options and positional parameters

74488 **SYNOPSIS**74489 set [-abCefhmnvux] [-o *option*] [*argument...*]74490 set [+abCefhmnvux] [+o *option*] [*argument...*]74491 set -- [*argument...*]

74492 set -o

74493 set +o

74494 **DESCRIPTION**

74495 If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables  
 74496 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the  
 74497 format:

74498 "%s=%s\n", &lt;name&gt;, &lt;value&gt;

74499 The *value* string shall be written with appropriate quoting; see the description of shell quoting in  
 74500 Section 2.2 (on page 2298). The output shall be suitable for reinput to the shell, setting or  
 74501 resetting, as far as possible, the variables that are currently set; read-only variables cannot be  
 74502 reset.

74503 When options are specified, they shall set or unset attributes of the shell, as described below.  
 74504 When *arguments* are specified, they cause positional parameters to be set or unset, as described  
 74505 below. Setting or unsetting attributes and positional parameters are not necessarily related  
 74506 actions, but they can be combined in a single invocation of *set*.

74507 The *set* special built-in shall support XBD Section 12.2 (on page 215) except that options can be  
 74508 specified with either a leading <hyphen> (meaning enable the option) or <plus-sign> (meaning  
 74509 disable it) unless otherwise specified.

74510 Implementations shall support the options in the following list in both their <hyphen> and  
 74511 <plus-sign> forms. These options can also be specified as options to *sh*.

74512 **-a** When this option is on, the *export* attribute shall be set for each variable to which an  
 74513 assignment is performed; see XBD Section 4.22 (on page 118). If the assignment precedes a  
 74514 utility name in a command, the *export* attribute shall not persist in the current execution  
 74515 environment after the utility completes, with the exception that preceding one of the special  
 74516 built-in utilities causes the *export* attribute to persist after the built-in has completed. If the  
 74517 assignment does not precede a utility name in the command, or if the assignment is a result  
 74518 of the operation of the *getopts* or *read* utilities, the *export* attribute shall persist until the  
 74519 variable is unset.

74520 **-b** This option shall be supported if the implementation supports the User Portability Utilities  
 74521 option. It shall cause the shell to notify the user asynchronously of background job  
 74522 completions. The following message is written to standard error:

74523 "[%d]c %s%s\n", &lt;job-number&gt;, &lt;current&gt;, &lt;status&gt;, &lt;job-name&gt;

74524 where the fields shall be as follows:

74525 <current> The character '+' identifies the job that would be used as a default for  
 74526 the *fg* or *bg* utilities; this job can also be specified using the *job\_id* "%+" or  
 74527 "%%". The character '-' identifies the job that would become the default  
 74528 if the current default job were to exit; this job can also be specified using  
 74529 the *job\_id* "%-". For other jobs, this field is a <space>. At most one job

- 74530 can be identified with '+' and at most one job can be identified with '-'.  
 74531 If there is any suspended job, then the current job shall be a suspended  
 74532 job. If there are at least two suspended jobs, then the previous job also  
 74533 shall be a suspended job.
- 74534 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*,  
 74535 and *kill* utilities. Using these utilities, the job can be identified by prefixing  
 74536 the job number with '%'.  
 74537 <status> Unspecified.  
 74538 <job-name> Unspecified.
- 74539 When the shell notifies the user a job has been completed, it may remove the job's process  
 74540 ID from the list of those known in the current shell execution environment; see [Section](#)  
 74541 [2.9.3.1](#) (on page 2319). Asynchronous notification shall not be enabled by default.
- 74542 -C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection  
 74543 operator (see [Section 2.7.2](#), on page 2313); the '>|' redirection operator shall override this  
 74544 *noclobber* option for an individual file.
- 74545 -e When this option is on, if a simple command fails for any of the reasons listed in [Section](#)  
 74546 [2.8.1](#) (on page 2315) or returns an exit status value >0, and is not part of the compound list  
 74547 following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a  
 74548 pipeline preceded by the ! reserved word, then the shell shall immediately exit.
- 74549 -f The shell shall disable pathname expansion.
- 74550 -h Locate and remember utilities invoked by functions as those functions are defined (the  
 74551 utilities are normally located when the function is executed).
- 74552 -m This option shall be supported if the implementation supports the User Portability Utilities  
 74553 option. All jobs shall be run in their own process groups. Immediately before the shell issues  
 74554 a prompt after completion of the background job, a message reporting the exit status of the  
 74555 background job shall be written to standard error. If a foreground job stops, the shell shall  
 74556 write a message to standard error to that effect, formatted as described by the *jobs* utility. In  
 74557 addition, if a job changes status other than exiting (for example, if it stops for input or  
 74558 output or is stopped by a SIGSTOP signal), the shell shall write a similar message  
 74559 immediately prior to writing the next prompt. This option is enabled by default for  
 74560 interactive shells.
- 74561 -n The shell shall read commands but does not execute them; this can be used to check for  
 74562 shell script syntax errors. An interactive shell may ignore this option.
- 74563 -o Write the current settings of the options to standard output in an unspecified format.
- 74564 +o Write the current option settings to standard output in a format that is suitable for reinput  
 74565 to the shell as commands that achieve the same options settings.
- 74566 -o *option*  
 74567 This option is supported if the system supports the User Portability Utilities option. It shall  
 74568 set various options, many of which shall be equivalent to the single option letters. The  
 74569 following values of *option* shall be supported:
- 74570 *allexport* Equivalent to -a.  
 74571 *errexit* Equivalent to -e.

74572	<i>ignoreeof</i>	Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>-D is entered. A user shall explicitly <i>exit</i> to leave the interactive shell.
74573		
74574		
74575	<i>monitor</i>	Equivalent to <b>-m</b> . This option is supported if the system supports the User Portability Utilities option.
74576		
74577	<i>noclobber</i>	Equivalent to <b>-C</b> (uppercase C).
74578	<i>noglob</i>	Equivalent to <b>-f</b> .
74579	<i>noexec</i>	Equivalent to <b>-n</b> .
74580	<i>nolog</i>	Prevent the entry of function definitions into the command history; see <a href="#">Command History List</a> (on page 3167).
74581		
74582	<i>notify</i>	Equivalent to <b>-b</b> .
74583	<i>nounset</i>	Equivalent to <b>-u</b> .
74584	<i>verbose</i>	Equivalent to <b>-v</b> .
74585	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension.
74586		
74587		
74588		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
74589	<i>xtrace</i>	Equivalent to <b>-x</b> .
74590	<b>-u</b>	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.
74591		
74592	<b>-v</b>	The shell shall write its input to standard error as it is read.
74593	<b>-x</b>	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
74594		
74595		
74596		The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see <i>sh</i> .
74597		
74598		The remaining arguments shall be assigned in order to the positional parameters. The special parameter '#' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
74599		
74600		
74601		If the first argument is <b>'-'</b> , the results are unspecified.
74602		The special argument <b>"--"</b> immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with <b>'+'</b> or <b>'-'</b> , or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set --</i> without <i>argument</i> shall unset all positional parameters and set the special parameter '#' to zero.
74603		
74604		
74605		
74606	<b>OPTIONS</b>	
74607		See the DESCRIPTION.
74608	<b>OPERANDS</b>	
74609		See the DESCRIPTION.

**set**74610 **STDIN**

74611 Not used.

74612 **INPUT FILES**

74613 None.

74614 **ENVIRONMENT VARIABLES**

74615 None.

74616 **ASYNCHRONOUS EVENTS**

74617 Default.

74618 **STDOUT**

74619 See the DESCRIPTION.

74620 **STDERR**

74621 The standard error shall be used only for diagnostic messages.

74622 **OUTPUT FILES**

74623 None.

74624 **EXTENDED DESCRIPTION**

74625 None.

74626 **EXIT STATUS**

74627 Zero.

74628 **CONSEQUENCES OF ERRORS**

74629 Default.

74630 **APPLICATION USAGE**

74631 None.

74632 **EXAMPLES**

74633 Write out all variables and their values:

74634 `set`

74635 Set \$1, \$2, and \$3 and set "\$#" to 3:

74636 `set c a b`74637 Turn on the `-x` and `-v` options:74638 `set -xv`

74639 Unset all positional parameters:

74640 `set --`74641 Set \$1 to the value of `x`, even if it begins with `'-'` or `'+'`:74642 `set -- "$x"`74643 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:74644 `set -- $x`74645 **RATIONALE**

74646 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the  
 74647 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether  
 74648 the `set --` form might be misinterpreted as being equivalent to `set` without any options or  
 74649 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`

74650 -- only unsets parameters if there is at least one argument; the only way to unset all parameters  
 74651 is to use *shift*. Using the KornShell version should not affect System V scripts because there  
 74652 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

```
74653 set -- "$@"
```

74654 and there were in fact no arguments resulting from "\$@", unsetting the parameters would have  
 74655 no result.

74656 The *set +* form in early proposals was omitted as being an unnecessary duplication of *set* alone  
 74657 and not widespread historical practice.

74658 The *noclobber* option was changed to allow *set -C* as well as the *set -o noclobber* option. The  
 74659 single-letter version was added so that the historical "\$-" paradigm would not be broken; see  
 74660 [Section 2.5.2](#) (on page 2302).

74661 The *-h* flag is related to command name hashing. See *hash* (on page 2788).

74662 The following *set* flags were omitted intentionally with the following rationale:

74663 **-k** The *-k* flag was originally added by the author of the Bourne shell to make it easier for  
 74664 users of pre-release versions of the shell. In early versions of the Bourne shell the construct  
 74665 *set name=value* had to be used to assign values to shell variables. The problem with *-k* is  
 74666 that the behavior affects parsing, virtually precluding writing any compilers. To explain the  
 74667 behavior of *-k*, it is necessary to describe the parsing algorithm, which is implementation-  
 74668 defined. For example:

```
74669 set -k; echo name=value
```

74670 and:

```
74671 set -k  
74672 echo name=value
```

74673 behave differently. The interaction with functions is even more complex. What is more, the  
 74674 *-k* flag is never needed, since the command line could have been reordered.

74675 **-t** The *-t* flag is hard to specify and almost never used. The only known use could be done  
 74676 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page  
 74677 says that it exits after reading and executing one command. What is one command? If the  
 74678 input is *date:date*, *sh* executes both *date* commands while *ksh* does only the first.

74679 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion  
 74680 was that the *unset* utility should be used to unset options instead of using the non-*getopt*(-)-able  
 74681 *+option* syntax. However, the conclusion was reached that the historical practice of using *+option*  
 74682 was satisfactory and that there was no compelling reason to modify such widespread historical  
 74683 practice.

74684 The *-o* option was adopted from the KornShell to address user needs. In addition to its  
 74685 generally friendly interface, *-o* is needed to provide the *vi* command line editing mode, for  
 74686 which historical practice yields no single-letter option name. (Although it might have been  
 74687 possible to invent such a letter, it was recognized that other editing modes would be developed  
 74688 and *-o* provides ample name space for describing such extensions.)

74689 Historical implementations are inconsistent in the format used for *-o* option status reporting.  
 74690 The *+o* format without an option-argument was added to allow portable access to the options  
 74691 that can be saved and then later restored using, for instance, a dot script.

74692 Historically, *sh* did trace the command *set +x*, but *ksh* did not.

- 74693 The *ignoreof* setting prevents accidental logouts when the end-of-file character (typically  
74694 <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.
- 74695 The *set -m* option was added to apply only to the UPE because it applies primarily to interactive  
74696 use, not shell script applications.
- 74697 The ability to do asynchronous notification became available in the 1988 version of the  
74698 KornShell. To have it occur, the user had to issue the command:
- 74699 `trap "jobs -n" CLD`
- 74700 The C shell provides two different levels of an asynchronous notification capability. The  
74701 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it  
74702 notifies the user immediately of background job completions. When unset, this capability is  
74703 turned off.
- 74704 The other notification ability comes through the built-in utility *notify*. The syntax is:
- 74705 `notify [%job ... ]`
- 74706 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when  
74707 the state of the current job changes. If given operands, *notify* asynchronously informs the user of  
74708 changes in the states of the specified jobs.
- 74709 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,  
74710 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX  
74711 environment variable name).
- 74712 The *set -b* option was selected as a compromise.
- 74713 The *notify* built-in was considered to have more functionality than was required for simple  
74714 asynchronous notification.
- 74715 **FUTURE DIRECTIONS**
- 74716 None.
- 74717 **SEE ALSO**
- 74718 [Section 2.14](#) (on page 2334), [hash](#)
- 74719 [XBD Section 4.22](#) (on page 118), [Section 12.2](#) (on page 215)
- 74720 **CHANGE HISTORY**
- 74721 **Issue 6**
- 74722 The obsolescent *set* command name followed by ' - ' has been removed.
- 74723 The following new requirements on POSIX implementations derive from alignment with the  
74724 Single UNIX Specification:
- 74725 • The *nolog* option is added to *set -o*.
- 74726 IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes  
74727 into account the description of the option.
- 74728 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74729 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74730 behavior is intended.
- 74731 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square  
74732 brackets in the example in RATIONALE to be in bold, which is the typeface used for optional  
74733 items.

74734 **Issue 7**

74735 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
74736 argument is ' -' .

74737 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74738 XSI shading is removed from the -h functionality.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**shift***Shell Command Language*74739 **NAME**

74740 shift — shift positional parameters

74741 **SYNOPSIS**74742 shift [*n*]74743 **DESCRIPTION**

74744 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of  
 74745 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The  
 74746 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the  
 74747 parameter '#' is updated to reflect the new number of positional parameters.

74748 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special  
 74749 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special  
 74750 parameters are not changed.

74751 **OPTIONS**

74752 None.

74753 **OPERANDS**

74754 See the DESCRIPTION.

74755 **STDIN**

74756 Not used.

74757 **INPUT FILES**

74758 None.

74759 **ENVIRONMENT VARIABLES**

74760 None.

74761 **ASYNCHRONOUS EVENTS**

74762 Default.

74763 **STDOUT**

74764 Not used.

74765 **STDERR**

74766 The standard error shall be used only for diagnostic messages.

74767 **OUTPUT FILES**

74768 None.

74769 **EXTENDED DESCRIPTION**

74770 None.

74771 **EXIT STATUS**74772 The exit status is >0 if *n*>\$#; otherwise, it is zero.74773 **CONSEQUENCES OF ERRORS**

74774 Default.

74775 **APPLICATION USAGE**

74776 None.

74777 **EXAMPLES**

74778 \$ set a b c d e

74779 \$ shift 2

74780 \$ echo \$\*

74781 c d e

74782 **RATIONALE**

74783 None.

74784 **FUTURE DIRECTIONS**

74785 None.

74786 **SEE ALSO**74787 [Section 2.14](#) (on page 2334)74788 **CHANGE HISTORY**74789 **Issue 6**

74790 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74791 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74792 behavior is intended.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**times**

Shell Command Language

74793 **NAME**74794 `times` — write process times74795 **SYNOPSIS**74796 `times`74797 **DESCRIPTION**74798 The *times* utility shall write the accumulated user and system times for the shell and for all of its  
74799 child processes, in the following POSIX locale format:74800 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,  
74801 <shell user seconds>, <shell system minutes>,  
74802 <shell system seconds>, <children user minutes>,  
74803 <children user seconds>, <children system minutes>,  
74804 <children system seconds>74805 The four pairs of times shall correspond to the members of the <sys/times.h> `tms` structure  
74806 (defined in XBD Chapter 13, on page 219) as returned by `times()`: `tms_utime`, `tms_stime`,  
74807 `tms_cutime`, and `tms_cstime`, respectively.74808 **OPTIONS**

74809 None.

74810 **OPERANDS**

74811 None.

74812 **STDIN**

74813 Not used.

74814 **INPUT FILES**

74815 None.

74816 **ENVIRONMENT VARIABLES**

74817 None.

74818 **ASYNCHRONOUS EVENTS**

74819 Default.

74820 **STDOUT**

74821 See the DESCRIPTION.

74822 **STDERR**

74823 The standard error shall be used only for diagnostic messages.

74824 **OUTPUT FILES**

74825 None.

74826 **EXTENDED DESCRIPTION**

74827 None.

74828 **EXIT STATUS**

74829 Zero.

74830 **CONSEQUENCES OF ERRORS**

74831 Default.

74832 **APPLICATION USAGE**

74833 None.

74834 **EXAMPLES**74835 `$ times`74836 `0m0.43s 0m1.11s`74837 `8m44.18s 1m43.23s`74838 **RATIONALE**74839 The *times* special built-in from the Single UNIX Specification is now required for all conforming  
74840 shells.74841 **FUTURE DIRECTIONS**

74842 None.

74843 **SEE ALSO**74844 [Section 2.14](#) (on page 2334)74845 XBD [<sys/times.h>](#)74846 **CHANGE HISTORY**74847 **Issue 6**74848 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the  
74849 DESCRIPTION from: "Write the accumulated user and system times for the shell and for all of  
74850 its child processes ..." to: "The *times* utility shall write the accumulated user and system times for  
74851 the shell and for all of its child processes ...".

**trap**

Shell Command Language

74852 **NAME**

74853 trap — trap signals

74854 **SYNOPSIS**

```
74855 trap n [condition...]
74856 trap [action condition...]
```

74857 **DESCRIPTION**

74858 If the first operand is an unsigned decimal integer, the shell shall treat all operands as  
 74859 conditions, and shall reset each condition to the default value. Otherwise, if there are operands,  
 74860 the first is treated as an action and the remaining as conditions.

74861 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (" "), the  
 74862 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read  
 74863 and executed by the shell when one of the corresponding conditions arises. The action of *trap*  
 74864 shall override a previous action (either default action or one explicitly set). The value of "\$?"  
 74865 after the *trap* action completes shall be the value it had before *trap* was invoked.

74866 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,  
 74867 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in  
 74868 XBD Chapter 13 (on page 219); for example, HUP, INT, QUIT, TERM. Implementations may  
 74869 permit names with the SIG prefix or ignore case in signal names as an extension. Setting a trap  
 74870 for SIGKILL or SIGSTOP produces undefined results.

74871 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment  
 74872 immediately after the last command executed before the *trap* on EXIT was taken.

74873 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

```
74874 eval action
```

74875 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although  
 74876 no error need be reported when attempting to do so. An interactive shell may reset or catch  
 74877 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed  
 74878 with another *trap* command.

74879 When a subshell is entered, traps that are not being ignored are set to the default actions. This  
 74880 does not imply that the *trap* command cannot be used within the subshell to set new traps.

74881 The *trap* command with no arguments shall write to standard output a list of commands  
 74882 associated with each condition. The format shall be:

```
74883 "trap -- %s %s ...\n", <action>, <condition> ...
```

74884 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 74885 reinput to the shell as commands that achieve the same trapping results. For example:

```
74886 save_traps=$(trap)
74887 ...
74888 eval "$save_traps"
```

74889 XSI  
 74890 XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to  
 the following signal names:

```
74891 1  SIGHUP
74892 2  SIGINT
74893 3  SIGQUIT
```

74894	6	SIGABRT
74895	9	SIGKILL
74896	14	SIGALRM
74897	15	SIGTERM

74898 The *trap* special built-in shall conform to XBD [Section 12.2](#) (on page 215).

#### 74899 OPTIONS

74900 None.

#### 74901 OPERANDS

74902 See the DESCRIPTION.

#### 74903 STDIN

74904 Not used.

#### 74905 INPUT FILES

74906 None.

#### 74907 ENVIRONMENT VARIABLES

74908 None.

#### 74909 ASYNCHRONOUS EVENTS

74910 Default.

#### 74911 STDOUT

74912 See the DESCRIPTION.

#### 74913 STDERR

74914 The standard error shall be used only for diagnostic messages.

#### 74915 OUTPUT FILES

74916 None.

#### 74917 EXTENDED DESCRIPTION

74918 None.

#### 74919 EXIT STATUS

74920 XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero  
 74921 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or  
 74922 numbers shall not be considered a syntax error and do not cause the shell to abort.

#### 74923 CONSEQUENCES OF ERRORS

74924 Default.

#### 74925 APPLICATION USAGE

74926 None.

#### 74927 EXAMPLES

74928 Write out a list of all traps and actions:

74929 trap

74930 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable  
 74931 executes when the shell terminates:

74932 trap '\$HOME/logout' EXIT

74933 or:

**trap**

74934 `trap '$HOME/logout' 0`

74935 Unset traps on INT, QUIT, TERM, and EXIT:

74936 `trap - INT QUIT TERM EXIT`

74937 **RATIONALE**

74938 Implementations may permit lowercase signal names as an extension. Implementations may  
74939 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*  
74940 utilities in this volume of POSIX.1-2008 are now consistent in their omission of the SIG prefix for  
74941 signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the signals  
74942 without prefixes.

74943 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but  
74944 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

74945 The output format is not historical practice. Since the output of historical *trap* commands is not  
74946 portable (because numeric signal values are not portable) and had to change to become so, an  
74947 opportunity was taken to format the output in a way that a shell script could use to save and  
74948 then later reuse a trap if it wanted.

74949 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is  
74950 allowable as an extension, but was not mandated, as other shells have not used it.

74951 The text about the environment for the EXIT trap invalidates the behavior of some historical  
74952 versions of interactive shells which, for example, close the standard input before executing a  
74953 trap on 0. For example, in some historical interactive shell sessions the following trap on 0  
74954 would always print "--":

74955 `trap 'read foo; echo "$foo"' 0`

74956 The command:

74957 `trap '$cmd' 0`

74958 causes the contents of the shell variable *cmd* to be executed as a command when the shell exits.  
74959 Using double-quotes instead of single-quotes might have unexpected behavior, since in theory  
74960 the value of *cmd* might be a decimal integer which would be treated as a condition, not an  
74961 action; or *cmd* might begin with '-'. Also, using double-quotes will cause the value of *cmd* to be  
74962 expanded twice, once when *trap* is executed, and once when the condition arises.

74963 **FUTURE DIRECTIONS**

74964 None.

74965 **SEE ALSO**

74966 [Section 2.14](#) (on page 2334)

74967 [XBD Section 12.2](#) (on page 215), [<signal.h>](#)

74968 **CHANGE HISTORY**

74969 **Issue 6**

74970 XSI-conforming implementations provide the mapping of signal names to numbers given above  
74971 (previously this had been marked obsolescent). Other implementations need not provide this  
74972 optional mapping.

74973 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
74974 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
74975 behavior is intended.

74976 **Issue 7**

74977 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74978 Austin Group Interpretation 1003.1-2001 #116 is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**unset**74979 **NAME**

74980           unset — unset values and attributes of variables and functions

74981 **SYNOPSIS**74982           unset [-fv] *name*...74983 **DESCRIPTION**74984           Each variable or function specified by *name* shall be unset.74985           If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from  
74986           the environment. Read-only variables cannot be unset.74987           If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.74988           If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not  
74989           exist, it is unspecified whether a function by that name, if any, shall be unset.74990           Unsetting a variable or function that was not previously set shall not be considered an error and  
74991           does not cause the shell to abort.74992           The *unset* special built-in shall support XBD [Section 12.2](#) (on page 215).

74993           Note that:

74994           VARIABLE=

74995           is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to ". Also, the  
74996           variables that can be *unset* should not be misinterpreted to include the special parameters (see  
74997           [Section 2.5.2](#), on page 2302).74998 **OPTIONS**

74999           See the DESCRIPTION.

75000 **OPERANDS**

75001           See the DESCRIPTION.

75002 **STDIN**

75003           Not used.

75004 **INPUT FILES**

75005           None.

75006 **ENVIRONMENT VARIABLES**

75007           None.

75008 **ASYNCHRONOUS EVENTS**

75009           Default.

75010 **STDOUT**

75011           Not used.

75012 **STDERR**

75013           The standard error shall be used only for diagnostic messages.

75014 **OUTPUT FILES**

75015           None.

75016 **EXTENDED DESCRIPTION**

75017           None.

75018 **EXIT STATUS**75019           0 All *name* operands were successfully unset.75020           >0 At least one *name* could not be unset.75021 **CONSEQUENCES OF ERRORS**

75022           Default.

75023 **APPLICATION USAGE**

75024           None.

75025 **EXAMPLES**75026           Unset *VISUAL* variable:

75027           unset -v VISUAL

75028           Unset the functions **foo** and **bar**:

75029           unset -f foo bar

75030 **RATIONALE**75031           Consideration was given to omitting the `-f` option in favor of an *unfunction* utility, but the  
75032           standard developers decided to retain historical practice.75033           The `-v` option was introduced because System V historically used one name space for both  
75034           variables and functions. When *unset* is used without options, System V historically unset either a  
75035           function or a variable, and there was no confusion about which one was intended. A portable  
75036           POSIX application can use *unset* without an option to unset a variable, but not a function; the `-f`  
75037           option must be used.75038 **FUTURE DIRECTIONS**

75039           None.

75040 **SEE ALSO**75041           [Section 2.14](#) (on page 2334)75042           XBD [Section 12.2](#) (on page 215)75043 **CHANGE HISTORY**75044 **Issue 6**75045           IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
75046           sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
75047           behavior is intended.75048 **Issue 7**

75049           SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

75050

Chapter 3

75051

**Batch Environment Services**

75052 OB BE

This chapter describes the services and utilities that shall be implemented on all systems that claim conformance to the Batch Environment Services and Utilities option. The functionality described in this section shall be provided on implementations that support the Batch Environment Services and Utilities option (and the rest of this section is not further shaded for this option).

75055

75056

Note that the Batch Environment Services and Utilities option is marked obsolescent in Issue 7.

75058

**3.1 General Concepts**

75059

**3.1.1 Batch Client-Server Interaction**

75060

Batch jobs are created and managed by batch servers. A batch client interacts with a batch server to access batch services on behalf of the user. In order to use batch services, a user must have access to a batch client.

75061

75062

75063

A batch server is a computational entity, such as a daemon process, that provides batch services. Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

75064

75065

The batch utilities described in this volume of POSIX.1-2008 (and listed in [Table 3-1](#)) are clients of batch services; they allow users to perform actions on the job such as creating, modifying, and deleting batch jobs from a shell command line. Although these batch utilities may be said to accomplish certain services, they actually obtain services on behalf of a user by means of requests to batch servers.

75066

75067

75068

75069

75070

**Table 3-1** Batch Utilities

75071

*qalter*    *qmove*    *qrls*    *qstat*

75072

*qdel*    *qmsg*    *qselect*    *qsub*

75073

*qhold*    *qrerun*    *qsig*

75074

Client-server interaction takes place by means of the batch requests defined in this chapter.

75075

Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

75076

75077

75078

75079 **3.1.2 Batch Queues**

75080 Two types of batch queue are described: routing queues and execution queues. When a batch job  
75081 is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing  
75082 queues under the following conditions:

- 75083 • The batch job has been routed to another queue.
- 75084 • The batch job has been deleted from the batch queue.
- 75085 • The batch job has been aborted.

75086 When a batch job is placed in an execution queue, it is a candidate for execution.

75087 A batch job is removed from an execution queue under the following conditions:

- 75088 • The batch job has been executed and exited.
- 75089 • The batch job has been aborted.
- 75090 • The batch job has been deleted from the batch queue.
- 75091 • The batch job has been moved to another queue.

75092 Access to a batch queue is limited to the batch server that manages the batch queue. Clients  
75093 never access a batch queue or a batch job directly, either to read or write information; all client  
75094 access to batch queues or jobs takes place through batch servers.

75095 **3.1.3 Batch Job Creation**

75096 When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier  
75097 to the job. A batch job identifier consists of both a sequence number that is unique among the  
75098 sequence numbers issued by that server and the name of the server. Since the batch server name  
75099 is unique within a name space, the job identifier is likewise unique within the name space.

75100 The batch server that creates a batch job shall return the batch server-assigned job identifier to  
75101 the client that requested the job creation. If the batch server routes or moves the job to another  
75102 server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job  
75103 shall never change.

75104 **3.1.4 Batch Job Tracking**

75105 Since a batch job may be moved after creation, the batch server name component of the job  
75106 identifier need not indicate the location of the job. An implementation may provide a batch job  
75107 tracking mechanism, in which case the user generally does not need to know the location of the  
75108 job. However, an implementation need not provide a batch job tracking mechanism, in which  
75109 case the user must find routed jobs by probing the possible destinations.

75110 **3.1.5 Batch Job Routing**

75111 To route a batch job, a batch server either moves the job to some other queue that is managed by  
75112 the batch server, or requests that some other batch server accept the job.

75113 Each routing queue has one or more queues to which it can route batch jobs. The batch server  
75114 administrator creates routing queues.

75115 A batch server may route a batch job from a routing queue to another routing queue. Batch  
75116 servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a  
75117 batch server routes jobs from the routing queues that it manages. The algorithm by which a  
75118 batch server selects a batch queue to which to route a batch job is implementation-defined.

75119 A batch job need not be eligible for routing to all the batch queues fed by the routing queue from  
75120 which it is routed. A batch server that has been asked to accept the job may reject the request if  
75121 the job requires resources that are unavailable to that batch server, or if the client is not  
75122 authorized to access the batch server.

75123 Batch servers may route high-priority jobs before low-priority jobs, but, on other than  
75124 overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a  
75125 routing queue reject requests to accept the job for reasons that are permanent, the batch server  
75126 that manages the job shall abort the job. If all or some rejections are temporary, the batch server  
75127 should try to route the job again at some later point.

75128 The reasons for rejecting a batch job are implementation-defined.

75129 The reasons for which the routing should be retried later and the reasons for which the job  
75130 should be aborted are also implementation-defined.

75131 **3.1.6 Batch Job Execution**

75132 To execute a batch job is to create a session leader (a process) that runs the shell program  
75133 indicated by the *Shell\_Path* attribute of the job. The script shall be passed to the program as its  
75134 standard input. An implementation may pass the script to the program by other  
75135 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
75136 the RUNNING state. The primary program that is executed by a batch job is typically, though  
75137 not necessarily, a shell program.

75138 A batch server shall execute eligible jobs as a deferred service—no client request is necessary  
75139 once the batch job is created and eligible. However, the attributes of a batch job, such as the job  
75140 hold type, may render the job ineligible. A batch server shall scan the execution queues that it  
75141 manages for jobs that are eligible for execution. The algorithm by which the batch server selects  
75142 eligible jobs for execution is implementation-defined.

75143 As part of creating the process for the batch job, the batch server shall open the standard output  
75144 and standard error streams of the session.

75145 The attributes of a batch job may indicate that the batch server executing the job shall send mail  
75146 to a list of users at the time it begins execution of the job.

75147 **3.1.7 Batch Job Exit**

75148 When the session leader of an executing job terminates, the job exits. As part of exiting a batch  
 75149 job, the batch server that manages the job shall remove the job from the batch queue in which it  
 75150 resides. The server shall transfer output files of the job to a location described by the attributes of  
 75151 the job.

75152 The attributes of a batch job may indicate that the batch server managing the job shall send mail  
 75153 to a list of users at the time the job exits.

75154 **3.1.8 Batch Job Abort**

75155 A batch server shall abort jobs for which a required deferred service cannot be performed. The  
 75156 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a  
 75157 list of users at the time it aborts the job.

75158 **3.1.9 Batch Authorization**

75159 Clients, such as the batch environment utilities (marked BE), access batch services by means of  
 75160 requests to one or more batch servers. To acquire the services of any given batch server, the user  
 75161 identifier under which the client runs must be authorized to use that batch server.

75162 The user with an associated user name that creates a batch job shall own the job and can perform  
 75163 actions such as read, modify, delete, and move.

75164 A user identifier of the same value at a different host need not be the same user. For example,  
 75165 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at  
 75166 host **beta**. Likewise, the same person may have access to different user names on different hosts.

75167 An implementation may optionally provide an authorization mechanism that permits one user  
 75168 name to access jobs under another user name.

75169 A process on a client host may be authorized to run processes under multiple user names at a  
 75170 batch server host. Where appropriate, the utilities defined in this volume of POSIX.1-2008  
 75171 provide a means for a user to choose from among such user names when creating or modifying  
 75172 a batch job.

75173 **3.1.10 Batch Administration**

75174 The processing of a batch job by a batch server is affected by the attributes of the job. The  
 75175 processing of a batch job may also be affected by the attributes of the batch queue in which the  
 75176 job resides and by the status of the batch server that manages the job. See also XBD [Chapter 3](#)  
 75177 (on page 33) for batch definitions.

75178 **3.1.11 Batch Notification**

75179 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*  
 75180 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events  
 75181 by sending mail to the user that owns the job, or to other designated users.

75182 **3.2 Batch Services**

75183 The presence of Batch Environment Services and Utilities option services is indicated by the  
 75184 configuration variable `POSIX2_PBS`. A conforming batch server provides services as defined in  
 75185 this section.

75186 A batch server shall provide batch services in two ways:

- 75187 1. The batch server provides a service at the request of a client.
- 75188 2. The batch server provides a deferred service as a result of a change in conditions  
 75189 monitored by the batch server.

75190 If a batch server cannot complete a request, it shall reject the request. If a batch server cannot  
 75191 complete a deferred service for a batch job, the batch server shall abort the batch job. [Table 3-2](#)  
 75192 is a summary of environment variables that shall be supported by an implementation of the batch  
 75193 server and utilities.

75194 **Table 3-2** Environment Variable Summary

Variable	Description
<code>PBS_DPREFIX</code>	Defines the directive prefix (see <i>qsub</i> )
<code>PBS_ENVIRONMENT</code>	Batch Job is batch or interactive (see <a href="#">Section 3.2.2.1</a> )
<code>PBS_JOBID</code>	The <i>job_identifier</i> attribute of job (see <a href="#">Section 3.2.3.8</a> )
<code>PBS_JOBNAME</code>	The <i>job_name</i> attribute of job (see <a href="#">Section 3.2.3.8</a> )
<code>PBS_O_HOME</code>	Defines the <i>HOME</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_HOST</code>	Defines the host name of the batch client (see <i>qsub</i> )
<code>PBS_O_LANG</code>	Defines the <i>LANG</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_LOGNAME</code>	Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_MAIL</code>	Defines the <i>MAIL</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_PATH</code>	Defines the <i>PATH</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_QUEUE</code>	Defines the submit queue of the batch client (see <i>qsub</i> )
<code>PBS_O_SHELL</code>	Defines the <i>SHELL</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_TZ</code>	Defines the <i>TZ</i> of the batch client (see <i>qsub</i> )
<code>PBS_O_WORKDIR</code>	Defines the working directory of the batch client (see <i>qsub</i> )
<code>PBS_QUEUE</code>	Defines the initial execution queue (see <a href="#">Section 3.2.2.1</a> )

75211 **3.2.1 Batch Job States**

75212 A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD,  
75213 WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests  
75214 that the batch server that manages the batch job can accept for the batch job. A batch server shall  
75215 change the state of a batch job either in response to service requests from clients or as a result of  
75216 deferred services, such as job execution or job routing.

75217 A batch job that is in the QUEUED state resides in a queue but is still pending either execution  
75218 or routing, depending on the queue type.

75219 A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED  
75220 state. A batch server that puts a batch job in an execution queue, but has not yet executed the  
75221 batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution  
75222 queue and is executing is defined to be in the RUNNING state. While a batch job is in the  
75223 RUNNING state, a session leader is associated with the batch job.

75224 A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute,  
75225 is defined to be in the HELD state.

75226 A batch job that is not held, but must wait until a future date and time before executing, is  
75227 defined to be in the WAITING state.

75228 When the session leader associated with a running job exits, the batch job shall be placed in the  
75229 EXITING state.

75230 A batch job for which the session leader has terminated is defined to be in the EXITING state,  
75231 and the batch server that manages such a batch job cannot accept job modification requests that  
75232 affect the batch job. While a batch job is in the EXITING state, the batch server that manages the  
75233 batch job is staging output files and notifying clients of job completion. Once a batch job has  
75234 exited, it no longer exists as an object managed by a batch server.

75235 A batch job that is being moved from a routing queue to another queue is defined to be in the  
75236 TRANSITING state.

75237 When a batch job in a routing queue has been selected to be moved to a new destination, then  
75238 the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the  
75239 batch server implementation.

75240 Batch jobs with either an *Execution\_Time* attribute value set in the future or a *Hold\_Types* attribute  
75241 of value not equal to NO\_HOLD, or both, may be routed or held in the routing queue. The  
75242 treatment of jobs with the *Execution\_Time* or *Hold\_Types* attributes in a routing queue is  
75243 implementation-defined.

75244 When a batch job in a routing queue has not been selected to be moved to a new destination and  
75245 the batch job has a *Hold\_Types* attribute value of other than NO\_HOLD, then the job should be in  
75246 the HELD state.

75247 **Note:** The effect of a hold upon a batch job in a routing queue is implementation-defined. The  
75248 implementation should use the state that matches whether the batch job can route with a hold  
75249 or not.

75250 When a batch job in a routing queue has not been selected to be moved to a new destination and  
75251 the batch job has:

- 75252 • A *Hold\_Types* attribute value of NO\_HOLD
- 75253 • An *Execution\_Time* attribute in the past

75254 then the batch job shall be in the QUEUED state.

75255 When a batch job in a routing queue has not been selected to be moved to a new destination and  
75256 the batch job has:

- 75257 • A *Hold\_Types* attribute value of NO\_HOLD
- 75258 • An *Execution\_Time* attribute in the future

75259 then the batch job may be in the WAITING state.

75260 **Note:** The effect of a future execution time upon a batch job in a routing queue is implementation-  
75261 defined. The implementation should use the state that matches whether the batch job can route  
75262 with a hold or not.

75263 [Table 3-3](#) describes the next state of a batch job, given the current state of the batch job and the  
75264 type of request. [Table 3-4](#) (on page 2383) describes the response of a batch server to a request,  
75265 given the current state of the batch job and the type of request.

### 75266 3.2.2 Deferred Batch Services

75267 This section describes the deferred services performed by batch servers: job execution, job  
75268 routing, job exit, job abort, and the rerunning of jobs after a restart.

#### 75269 3.2.2.1 Batch Job Execution

75270 To execute a batch job is to create a session leader (a process) that runs the shell program  
75271 indicated by the *Shell\_Path\_List* attribute of the batch job. The script is passed to the program as  
75272 its standard input. An implementation may pass the script to the program by other  
75273 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
75274 the RUNNING state.

75275 **Table 3-3** Next State Table

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	Q	e	e	e	e	e	e
Modify Batch Job Request	e	Q	R	H	W	e	T
Delete Batch Job Request	e	X	E	X	X	E	X
Batch Job Message Request	e	Q	R	H	W	E	T
Rerun Batch Job Request	e	e	Q	e	e	e	e
Signal Batch Job Request	e	e	R	H	W	e	e
Batch Job Status Request	e	Q	R	H	W	E	T
Batch Queue Status Request	X	Q	R	H	W	E	T
Server Status Request	X	Q	R	H	W	E	T
Select Batch Jobs Request	X	Q	R	H	W	E	T
Move Batch Job Request	e	Q	R	H	W	e	T
Hold Batch Job Request	e	H	R/H	H	H	e	T
Release Batch Job Request	e	Q	R	Q/W/H	W	e	T
Server Shutdown Request	X	Q	Q	H	W	E	T
Locate Batch Job Request	e	Q	R	H	W	E	T

75293 **Legend**

75294 X Nonexistent

75295 Q QUEUED

75296 R RUNNING

75297 H HELD

75298 W WAITING

75299 E EXITING

75300 T TRANSITING

75301 e Error

75302 A batch server that has an execution queue containing jobs is said to own the queue and manage  
 75303 the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in  
 75304 the execution queues owned by the batch server. The batch server shall schedule for execution  
 75305 those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling  
 75306 jobs is implementation-defined.

75307 A batch server that executes a batch job shall create, in the environment of the session leader of  
 75308 the batch job, an environment variable named *PBS\_ENVIRONMENT*, the value of which is the  
 75309 string *PBS\_BATCH* encoded in the portable character set.

75310 A batch server that executes a batch job shall create, in the environment of the session leader of  
 75311 the batch job, an environment variable named *PBS\_QUEUE*, the value of which is the name of  
 75312 the execution queue of the batch job encoded in the portable character set.

75313 To rerun a batch job is to requeue a batch job that is currently executing and then kill the session  
 75314 leader of the executing job by sending a SIGKILL prior to completion; see [Section 3.2.3.11](#) (on  
 75315 page 2395). A batch server that reruns a batch job shall append the standard output and  
 75316 standard error files of the batch job to the corresponding files of the previous execution, if they  
 75317 exist, with appropriate annotation. If either file does not exist, that file shall be created as in  
 75318 normal execution.

75319

Table 3-4 Results/Output Table

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	O	e	e	e	e	e	e
Modify Batch Job Request	e	O	e	O	O	e	e
Delete Batch Job Request	e	O	O	O	O	e	O
Batch Job Message Request	e	e	O	e	e	e	e
Rerun Batch Job Request	e	e	O	e	e	e	e
Signal Batch Job Request	e	e	O	e	e	e	e
Batch Job Status Request	e	O	O	O	O	O	O
Batch Queue Status Request	O	O	O	O	O	O	O
Server Status Request	O	O	O	O	O	O	O
Select Batch Job Request	e	O	O	O	O	O	O
Move Batch Job Request	e	O	O	O	O	e	e
Hold Batch Job Request	e	O	O	O	O	e	e
Release Batch Job Request	e	O	e	O	O	e	e
Server Shutdown Request	O	O	e	O	O	e	e
Locate Batch Job Request	e	O	O	O	O	O	O

75337 **Legend**

75338 O OK

75339 e Error message

75340 The execution of a batch job by a batch server shall be controlled by job, queue, and server  
75341 attributes, as defined in this section.

75342 **Account\_Name Attribute**

75343 Batch accounting is an optional feature of batch servers. If a batch server implements  
75344 accounting, the statements in this section apply and the configuration variable  
75345 POSIX2\_PBS\_ACCOUNTING shall be set to 1.

75346 A batch server that executes a batch job shall charge the account named in the *Account\_Name*  
75347 attribute of the batch job for resources consumed by the batch job.

75348 If the *Account\_Name* attribute of the batch job is absent from the batch job attribute list or is  
75349 altered while the batch job is in execution, the batch server action is implementation-defined.

75350 **Checkpoint Attribute**

75351 Batch checkpointing is an optional feature of batch servers. If a batch server implements  
75352 checkpointing, the statements in this section apply and the configuration variable  
75353 POSIX2\_PBS\_CHECKPOINT shall be set to 1.

75354 There are two attributes associated with the checkpointing feature: *Checkpoint* and  
75355 *Minimum\_Cpu\_Interval*. *Checkpoint* is a batch job attribute, while *Minimum\_Cpu\_Interval* is a  
75356 queue attribute. An implementation that does not support checkpointing shall support the  
75357 *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute  
75358 to other servers.

75359 The behavior of a batch server that executes a batch job for which the value of the *Checkpoint*  
75360 attribute is CHECKPOINT\_UNSPECIFIED is implementation-defined. A batch server that  
75361 executes a batch job for which the value of the *Checkpoint* attribute is NO\_CHECKPOINT shall

75362 not checkpoint the batch job.

75363 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
75364 CHECKPOINT\_AT\_SHUTDOWN shall checkpoint the batch job only when the batch server  
75365 accepts a request to shut down during the time when the batch job is in the RUNNING state.

75366 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
75367 CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL shall checkpoint the batch job at the interval  
75368 specified by the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job has been  
75369 selected. The *Minimum\_Cpu\_Interval* attribute shall be specified in units of CPU minutes.

75370 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an  
75371 unsigned integer shall checkpoint the batch job at an interval that is the value of either the  
75372 *Checkpoint* attribute, or the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job  
75373 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When  
75374 the *Minimum\_Cpu\_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall  
75375 write a warning message to the standard error stream of the batch job.

#### 75376 **Error\_Path Attribute**

75377 The *Error\_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When  
75378 the *Join\_Path* attribute of the batch job is set to the value FALSE and the *Keep\_Files* attribute of  
75379 the batch job does not contain the value KEEP\_STD\_ERROR, a batch server that executes a batch  
75380 job shall perform one of the following actions:

- 75381 • Set the standard error stream of the session leader of the batch job to the path described by  
75382 the value of the *Error\_Path* attribute of the batch job.
- 75383 • Buffer the standard error of the session leader of the batch job until completion of the batch  
75384 job, and when the batch job exits return the contents to the destination described by the  
75385 value of the *Error\_Path* attribute of the batch job.

75386 Applications shall not rely on having access to the standard error of a batch job prior to the  
75387 completion of the batch job.

75388 When the *Error\_Path* attribute does not specify a host name, then the batch server shall retain the  
75389 standard error of the batch job on the host of execution.

75390 When the *Error\_Path* attribute does specify a host name and the *Keep\_Files* attribute does not  
75391 contain the value KEEP\_STD\_ERROR, then the final destination of the standard error of the  
75392 batch job shall be on the host whose host name is specified.

75393 If the path indicated by the value of the *Error\_Path* attribute of the batch job is a relative path, the  
75394 batch server shall expand the path relative to the home directory of the user on the host to which  
75395 the file is being returned.

75396 When the batch server buffers the standard error of the batch job and the file cannot be opened  
75397 for write upon completion of the batch job, then the server shall place the standard error in an  
75398 implementation-defined location and notify the user of the location via mail. It shall be possible  
75399 for the user to process this mail using the *mailx* utility.

75400 If a batch server that does not buffer the standard error cannot open the standard error path of  
75401 the batch job for write access, then the batch server shall abort the batch job.

75402 **Execution\_Time Attribute**

75403 A batch server shall not execute a batch job before the time represented by the value of the  
75404 *Execution\_Time* attribute of the batch job. The *Execution\_Time* attribute is defined in seconds since  
75405 the Epoch.

75406 **Hold\_Types Attribute**

75407 A batch server shall support the following hold types:

- 75408 s Can be set or released by a user with at least a privilege level of batch administrator  
75409 (SYSTEM).
- 75410 o Can be set or released by a user with at least a privilege level of batch operator  
75411 (OPERATOR).
- 75412 u Can be set or released by the user with at least a privilege level of user, where the user is  
75413 defined in the *Job\_Owner* attribute (USER).
- 75414 n Indicates that none of the *Hold\_Types* attributes are set (NO\_HOLD).

75415 An implementation may define other hold types. Any additional hold types, how they are  
75416 specified, their internal representation, their behavior, and how they affect the behavior of other  
75417 utilities are implementation-defined.

75418 The value of the *Hold\_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u',  
75419 and any implementation-defined hold types), or 'n'.

75420 A batch server shall not execute a batch job if the *Hold\_Types* attribute of the batch job has a  
75421 value other than NO\_HOLD. If the *Hold\_Types* attribute of the batch job has a value other than  
75422 NO\_HOLD, the batch job shall be in the HELD state.

75423 **Job\_Owner Attribute**

75424 The *Job\_Owner* attribute consists of a pair of user name and host name values of the form:

75425 `username@hostname`

75426 A batch server that accepts a *Queue Batch Job Request* shall set the *Job\_Owner* attribute to a string  
75427 that is the `username@hostname` of the user who submitted the job.

75428 **Join\_Path Attribute**

75429 A batch server that executes a batch job for which the value of the *Join\_Path* attribute is TRUE  
75430 shall ignore the value of the *Error\_Path* attribute and merge the standard error of the batch job  
75431 with the standard output of the batch job.

75432 **Keep\_Files Attribute**

75433 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
75434 the value KEEP\_STD\_OUTPUT shall retain the standard output of the batch job on the host  
75435 where execution occurs. The standard output shall be retained in the home directory of the user  
75436 under whose user ID the batch job is executed and the filename shall be the default filename for  
75437 the standard output as defined under the `-o` option of the *qsub* utility. The *Output\_Path* attribute  
75438 is not modified.

75439 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
75440 the value KEEP\_STD\_ERROR shall retain the standard error of the batch job on the host where  
75441 execution occurs. The standard error shall be retained in the home directory of the user under  
75442 whose user ID the batch job is executed and the filename shall be the default filename for

75443 standard error as defined under the `-e` option of the `qsub` utility. The `Error_Path` attribute is not  
75444 modified.

75445 A batch server that executes a batch job for which the value of the `Keep_Files` attribute includes  
75446 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on  
75447 the host where execution occurs. These files (with implementation-defined names) shall be  
75448 retained in the home directory of the user under whose user identifier the batch job is executed.

#### 75449 **Mail\_Points and Mail\_Users Attributes**

75450 A batch server that executes a batch job for which one of the values of the `Mail_Points` attribute is  
75451 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the  
75452 `Mail_Users` attribute of the batch job.

75453 The mail message shall contain at least the batch job identifier, queue, and server at which the  
75454 batch job currently resides, and the `Job_Owner` attribute.

#### 75455 **Output\_Path Attribute**

75456 The `Output_Path` attribute of a running job cannot be changed by a `Modify Batch Job Request`.  
75457 When the `Keep_Files` attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a  
75458 batch server that executes a batch job shall either:

- 75459 • Set the standard output stream of the session leader of the batch job to the destination  
75460 described by the value of the `Output_Path` attribute of the batch job.

75461 or:

- 75462 • Buffer the standard output of the session leader of the batch job until completion of the  
75463 batch job, and when the batch job exits return the contents to the destination described by  
75464 the value of the `Output_Path` attribute of the batch job.

75465 When the `Output_Path` attribute does not specify a host name, then the batch server shall retain  
75466 the standard output of the batch job on the host of execution.

75467 When the `Keep_Files` attribute does not contain the value `KEEP_STD_OUTPUT` and the  
75468 `Output_Path` attribute does specify a host name, then the final destination of the standard output  
75469 of the batch job shall be on the host specified.

75470 If the path specified in the `Output_Path` attribute of the batch job is a relative path, the batch  
75471 server shall expand the path relative to the home directory of the user on the host to which the  
75472 file is being returned.

75473 Whether or not the batch server buffers the standard output of the batch job until completion of  
75474 the batch job is implementation-defined. Applications shall not rely on having access to the  
75475 standard output of a batch job prior to the completion of the batch job.

75476 When the batch server does buffer the standard output of the batch job and the file cannot be  
75477 opened for write upon completion of the batch job, then the batch server shall place the standard  
75478 output in an implementation-defined location and notify the user of the location via mail. It shall  
75479 be possible for the user to process this mail using the `mailx` utility.

75480 If a batch server that does not buffer the standard output cannot open the standard output path  
75481 of the batch job for write access, then the batch server shall abort the batch job.

75482 **Priority Attribute**

75483 A batch server implementation may choose to preferentially execute a batch job based on the  
 75484 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is  
 75485 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger  
 75486 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

75487 **Rerunable Attribute**

75488 A batch job that began execution but did not complete, because the batch server either shut  
 75489 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has  
 75490 the value TRUE.

75491 If a batch job, which was requeued after beginning execution but prior to completion, has a valid  
 75492 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted  
 75493 from the last valid checkpoint.

75494 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*  
 75495 attribute value of TRUE and was requeued after beginning execution but prior to completion,  
 75496 the batch server shall place the batch job into execution at the beginning of the job.

75497 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after  
 75498 beginning execution but prior to completion, and the batch job cannot be restarted from a  
 75499 checkpoint, then the batch server shall abort the batch job.

75500 **Resource\_List Attribute**

75501 A batch server that executes a batch job shall establish the resource limits of the session leader of  
 75502 the batch job according to the values of the *Resource\_List* attribute of the batch job. Resource  
 75503 limits shall be enforced by an implementation-defined method.

75504 **Shell\_Path\_List Attribute**

75505 The *Shell\_Path\_List* job attribute consists of a list of pairs of pathname and host name values. The  
 75506 host name component can be omitted, in which case the pathname serves as the default  
 75507 pathname when a batch server cannot find the name of the host on which it is running in the list.

75508 A batch server that executes a batch job shall select, from the value of the *Shell\_Path\_List*  
 75509 attribute of the batch job, a pathname where the shell to execute the batch job shall be found.  
 75510 The batch server shall select the pathname, in order of preference, according to the following  
 75511 methods:

- 75512 • Select the pathname that contains the name of the host on which the batch server is  
 75513 running.
- 75514 • Select the pathname for which the host name has been omitted.
- 75515 • Select the pathname for the login shell of the user under which the batch job is to execute.

75516 If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

75517 If the value of the selected pathname from the *Shell\_Path\_List* attribute of the batch job  
 75518 represents a partial path, the batch server shall expand the path relative to a path that is  
 75519 implementation-defined.

75520 The batch server that executes the batch job shall execute the program that was selected from the  
 75521 *Shell\_Path\_List* attribute of the batch job. The batch server shall pass the path to the script of the  
 75522 batch job as the first argument to the shell program.

75523 **User\_List Attribute**

75524 The *User\_List* job attribute consists of a list of pairs of user name and host name values. The host  
 75525 name component can be omitted, in which case the user name serves as a default when a batch  
 75526 server cannot find the name of the host on which it is running in the list.

75527 A batch server that executes a batch job shall select, from the value of the *User\_List* attribute of  
 75528 the batch job, a user name under which to create the session leader. The server shall select the  
 75529 user name, in order of preference, according to the following methods:

- 75530 • Select the user name of a value that contains the name of the host on which the batch  
 75531 server executes.
- 75532 • Select the user name of a value for which the host name has been omitted.
- 75533 • Select the user name from the *Job\_Owner* attribute of the batch job.

75534 **Variable\_List Attribute**

75535 A batch server that executes a batch job shall create, in the environment of the session leader of  
 75536 the batch job, each environment variable listed in the *Variable\_List* attribute of the batch job, and  
 75537 set the value of each such environment variable to that of the corresponding variable in the  
 75538 variable list.

75539 3.2.2.2 *Batch Job Routing*

75540 To route a batch job is to select a queue from a list and move the batch job to that queue.

75541 A batch server that has routing queues, which have been started, shall route the jobs in the  
 75542 routing queues owned by the batch server. A batch server may delay the routing of a batch job.  
 75543 The algorithm for selecting a batch job and the queue to which it will be routed is  
 75544 implementation-defined.

75545 When a routing queue has multiple possible destinations specified, then the precedence of the  
 75546 destinations is implementation-defined.

75547 A batch server that routes a batch job to a queue at another server shall move the batch job into  
 75548 the target queue with a *Queue Batch Job Request*.

75549 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the  
 75550 batch job or abort the batch job. A batch server that retries failed routings shall provide a means  
 75551 for the batch administrator to specify the number of retries and the minimum period of time  
 75552 between retries. The means by which an administrator specifies the number of retries and the  
 75553 delay between retries is implementation-defined. When the number of retries specified by the  
 75554 batch administrator has been exhausted, the batch server shall abort the batch job and perform  
 75555 the functions of *Batch Job Exit*; see Section 3.2.2.3.

75556 3.2.2.3 *Batch Job Exit*

75557 For each job in the EXITING state, the batch server that exited the batch job shall perform the  
 75558 following deferred services in the order specified:

- 75559 1. If buffering standard error, move that file into the location specified by the *Error\_Path*  
 75560 attribute of the batch job.
- 75561 2. If buffering standard output, move that file into the location specified by the *Output\_Path*  
 75562 attribute of the batch job.

75563 3. If the *Mail\_Points* attribute of the batch job includes MAIL\_AT\_EXIT, send mail to the  
 75564 users listed in the *Mail\_Users* attribute of the batch job. The mail message shall contain at  
 75565 least the batch job identifier, queue, and server at which the batch job currently resides,  
 75566 and the *Job\_Owner* attribute.

75567 4. Remove the batch job from the queue.

75568 If a batch server that buffers the standard error output cannot return the standard error file to  
 75569 the standard error path at the time the batch job exits, the batch server shall do one of the  
 75570 following:

- 75571 • Mail the standard error file to the batch job owner.
- 75572 • Save the standard error file and mail the location and name of the file where the standard  
 75573 error is stored to the batch job owner.
- 75574 • Save the standard error file and notify the user by other implementation-defined means.

75575 If a batch server that buffers the standard output cannot return the standard output file to the  
 75576 standard output path at the time the batch job exits, the batch server shall do one of the  
 75577 following:

- 75578 • Mail the standard output file to the batch job owner.
- 75579 • Save the standard output file and mail the location and name of the file where the standard  
 75580 output is stored to the batch job owner.
- 75581 • Save the standard output file and notify the user by other implementation-defined means.

75582 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

#### 75583 3.2.2.4 *Batch Server Restart*

75584 A batch server that has been either shutdown or terminated abnormally, and has returned to  
 75585 operation, is said to have “restarted”.

75586 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were  
 75587 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*  
 75588 attribute of the batch job has the value TRUE.

75589 Queues are defined to be non-volatile. A batch server shall store the content of queues that it  
 75590 controls in such a way that server and system shutdowns do not erase the content of the queues.

#### 75591 3.2.2.5 *Batch Job Abort*

75592 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

75593 A batch server that aborts a batch job shall perform the following services:

- 75594 • Delete the batch job from the queue in which it resides.
- 75595 • If the *Mail\_Points* attribute of the batch job includes the value MAIL\_AT\_ABORT, send  
 75596 mail to the users listed in the value of the *Mail\_Users* attribute of the job. The mail message  
 75597 shall contain at least the batch job identifier, queue, and server at which the batch job  
 75598 currently resides, the *Job\_Owner* attribute, and the reason for the abort.
- 75599 • If the batch job was in the RUNNING state, terminate the session leader of the executing  
 75600 job by sending the session leader a SIGKILL, place the batch job in the EXITING state, and  
 75601 perform the actions of *Batch Job Exit*.

75602 **3.2.3 Requested Batch Services**

75603 This section describes the services provided by batch servers in response to requests from  
 75604 clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type  
 75605 (deferred or not) and whether it is an optional function.

75606 **Table 3-5** Batch Services Summary

Batch Service	Deferred	Optional
<i>Batch Job Execution</i>	Yes	No
<i>Batch Job Routing</i>	Yes	No
<i>Batch Job Exit</i>	Yes	No
<i>Batch Server Restart</i>	Yes	No
<i>Batch Job Abort</i>	Yes	No
<i>Delete Batch Job Request</i>	No	No
<i>Hold Batch Job Request</i>	No	No
<i>Batch Job Message Request</i>	No	Yes
<i>Batch Job Status Request</i>	No	No
<i>Locate Batch Job Request</i>	No	Yes
<i>Modify Batch Job Request</i>	No	No
<i>Move Batch Job Request</i>	No	No
<i>Queue Batch Job Request</i>	No	No
<i>Batch Queue Status Request</i>	No	No
<i>Release Batch Job Request</i>	No	No
<i>Rerun Batch Job Request</i>	No	No
<i>Select Batch Jobs Request</i>	No	No
<i>Server Shutdown Request</i>	No	No
<i>Server Status Request</i>	No	No
<i>Signal Batch Job Request</i>	No	No
<i>Track Batch Job Request</i>	No	Yes

75629 If a request is rejected because the batch client is not authorized to perform the action, the batch  
 75630 server shall return the same status as when the batch job does not exist.

75631 **3.2.3.1 Delete Batch Job Request**

75632 A batch job is defined to have been deleted when it has been removed from the queue in which  
 75633 it resides and not instantiated in another queue. A client requests that the server that manages a  
 75634 batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

75635 A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- 75636 • The user of the batch client is not authorized to delete the designated job.
- 75637 • The designated job is not managed by the batch server.
- 75638 • The designated job is in a state inconsistent with the delete request.

75639 A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons.  
 75640 The method used to determine whether the user of a client is authorized to perform the  
 75641 requested action is implementation-defined.

75642 A batch server requested to delete a batch job shall delete the batch job if the batch job exists and  
 75643 is not in the EXITING state.

75644 A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the

75645 session leader of the batch job. It is implementation-defined whether additional signals are sent  
75646 to the session leader of the job prior to sending the SIGKILL signal.

75647 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the  
75648 EXITING state after it has killed the session leader of the batch job and shall perform the actions  
75649 of *Batch Job Exit*.

### 75650 3.2.3.2 *Hold Batch Job Request*

75651 A batch client can request that the batch server add one or more holds to a batch job. Such a  
75652 request is called a *Hold Batch Job Request*.

75653 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- 75654 • The batch server does not support one or more of the requested holds to be added to the  
75655 batch job.
- 75656 • The user of the batch client is not authorized to add one or more of the requested holds to  
75657 the batch job.
- 75658 • The batch server does not manage the specified job.
- 75659 • The designated job is in the EXITING state.

75660 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The  
75661 method used to determine whether the user of a client is authorized to perform the requested  
75662 action is implementation-defined.

75663 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall  
75664 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the  
75665 RUNNING state are implementation-defined.

75666 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*  
75667 *Batch Job Request*, that is not already present, to the value of the *Hold\_Types* attribute of the batch  
75668 job.

### 75669 3.2.3.3 *Batch Job Message Request*

75670 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports  
75671 *Batch Job Message Request*, the statements in this section apply and the configuration variable  
75672 POSIX2\_PBS\_MESSAGE shall be set to 1.

75673 A batch client can request that a batch server write a message into certain output files of a batch  
75674 job. Such a request is called a *Batch Job Message Request*.

75675 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- 75676 • The batch server does not support sending messages to jobs.
- 75677 • The user of the batch client is not authorized to post a message to the designated job.
- 75678 • The designated job does not exist on the batch server.
- 75679 • The designated job is not in the RUNNING state.

75680 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons.  
75681 The method used to determine whether the user of a client is authorized to perform the  
75682 requested action is implementation-defined.

75683 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch  
75684 client into the files indicated by the batch client.

*Batch Services**Batch Environment Services*75685 3.2.3.4 *Batch Job Status Request*

75686 A batch client can request that a batch server respond with the status and attributes of a batch  
75687 job. Such a request is called a *Batch Job Status Request*.

75688 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 75689 • The user of the batch client is not authorized to query the status of the designated job.
- 75690 • The designated job is not managed by the batch server.

75691 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons.  
75692 The method used to determine whether the user of a client is authorized to perform the  
75693 requested action is implementation-defined.

75694 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the  
75695 batch client.

75696 A batch server may return other information in response to a *Batch Job Status Request*.

75697 3.2.3.5 *Locate Batch Job Request*

75698 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports  
75699 *Locate Batch Job Request*, the statements in this section apply and the configuration variable  
75700 POSIX2\_PBS\_LOCATE shall be set to 1.

75701 A batch client can ask a batch server to respond with the location of a batch job that was created  
75702 by the batch server. Such a request is called a *Locate Batch Job Request*.

75703 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to  
75704 the batch client.

75705 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that  
75706 server.

75707 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by  
75708 that server; that is, for a batch job that is not in a queue owned by that server.

75709 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

75710 3.2.3.6 *Modify Batch Job Request*

75711 Batch clients modify (alter) the attributes of a batch job by making a request to the server that  
75712 manages the batch job. Such a request is called a *Modify Batch Job Request*.

75713 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 75714 • The user of the batch client is not authorized to make the requested modification to the  
75715 batch job.
- 75716 • The designated job is not managed by the batch server.
- 75717 • The requested modification is inconsistent with the state of the batch job.
- 75718 • An unrecognized resource is requested for a batch job in an execution queue.

75719 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons.  
75720 The method used to determine whether the user of a client is authorized to perform the  
75721 requested action is implementation-defined.

75722 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of  
75723 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the

75724 attributes of the batch job.

75725 If the servicing by a batch server of an otherwise valid request would result in no change, then  
75726 the batch server shall indicate successful completion of the request.

### 75727 3.2.3.7 Move Batch Job Request

75728 A batch client can request that a batch server move a batch job to another destination. Such a  
75729 request is called a *Move Batch Job Request*.

75730 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- 75731 • The user of the batch client is not authorized to remove the designated job from the queue  
75732 in which the batch job resides.
- 75733 • The user of the batch client is not authorized to move the designated job to the destination.
- 75734 • The designated job is not managed by the batch server.
- 75735 • The designated job is in the EXITING state.
- 75736 • The destination is inaccessible.

75737 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The  
75738 method used to determine whether the user of a client is authorized to perform the requested  
75739 action is implementation-defined.

75740 A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- 75741 • Queue the designated job at the destination.
- 75742 • Remove the designated job from the queue in which the batch job resides.

75743 If the destination resides on another batch server, the batch server shall queue the batch job at  
75744 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*  
75745 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job*  
75746 *Request* succeeds, the batch server shall remove the batch job from its queue.

75747 The batch server shall not modify any attributes of the batch job.

### 75748 3.2.3.8 Queue Batch Job Request

75749 A batch queue is controlled by one and only one batch server. A batch server is said to own the  
75750 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a  
75751 request is called a *Queue Batch Job Request*.

75752 A batch server requested to queue a batch job for which the queue is not specified shall select an  
75753 implementation-defined queue for the batch job. Such a queue is called the “default queue” of  
75754 the batch server. The implementation shall provide the means for a batch administrator to  
75755 specify the default queue. The queue, whether specified or defaulted, is called the “target  
75756 queue”.

75757 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 75758 • The client is not authorized to create a batch job in the target queue.
- 75759 • The request specifies a queue that does not exist on the batch server.
- 75760 • The target queue is an execution queue and the batch server cannot satisfy a resource  
75761 requirement of the batch job.

- 75762           • The target queue is an execution queue and an unrecognized resource is requested.
- 75763           • The target queue is an execution queue, the batch server does not support checkpointing,  
75764           and the value of the *Checkpoint* attribute of the batch job is not NO\_CHECKPOINT.
- 75765           • The job requires access to a user identifier that the batch client is not authorized to access.
- 75766           A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.
- 75767           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
75768           PBS\_O\_QUEUE value is missing from the value of the *Variable\_List* attribute of the batch job  
75769           shall add that variable to the list and set the value to the name of the target queue. Once set, no  
75770           server shall change the value of PBS\_O\_QUEUE, even if the batch job is moved to another  
75771           queue.
- 75772           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS\_JOBID  
75773           value is missing from the value of the *Variable\_List* attribute shall add that variable to the list and  
75774           set the value to the batch job identifier assigned by the server in the format:
- 75775           sequence\_number.server
- 75776           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
75777           PBS\_JOBNAME value is missing from the value of the *Variable\_List* attribute of the batch job  
75778           shall add that variable to the list and set the value to the *Job\_Name* attribute of the batch job.
- 75779   3.2.3.9   *Batch Queue Status Request*
- 75780           A batch client can request that a batch server respond with the status and attributes of a queue.  
75781           Such a request is called a *Batch Queue Status Request*.
- 75782           A batch server shall reject a *Batch Queue Status Request* if any of the following statements are  
75783           true:
- 75784           • The user of the batch client is not authorized to query the status of the designated queue.
- 75785           • The designated queue does not exist on the batch server.
- 75786           A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons.  
75787           The method used to determine whether the user of a client is authorized to perform the  
75788           requested action is implementation-defined.
- 75789           A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to  
75790           the batch client.
- 75791   3.2.3.10 *Release Batch Job Request*
- 75792           A batch client can request that the server remove one or more holds from a batch job. Such a  
75793           request is called a *Release Batch Job Request*.
- 75794           A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:
- 75795           • The user of the batch client is not authorized to remove one or more of the requested holds  
75796           from the batch job.
- 75797           • The batch server does not manage the specified job.
- 75798           A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons.  
75799           The method used to determine whether the user of a client is authorized to perform the  
75800           requested action is implementation-defined.
- 75801           A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the

75802 *Release Batch Job Request*, that is present, from the value of the *Hold\_Types* attribute of the batch  
75803 job.

#### 75804 3.2.3.11 *Rerun Batch Job Request*

75805 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible  
75806 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request  
75807 is called *Rerun Batch Job Request*.

75808 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 75809 • The user of the batch client is not authorized to rerun the designated job.
- 75810 • The *Rerunable* attribute of the designated job has the value FALSE.
- 75811 • The designated job is not in the RUNNING state.
- 75812 • The batch server does not manage the designated job.

75813 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons.  
75814 The method used to determine whether the user of a client is authorized to perform the  
75815 requested action is implementation-defined.

75816 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the  
75817 batch job.

75818 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 75819 • Requeue the batch job in the execution queue in which it was executing.
- 75820 • Send a SIGKILL signal to the process group of the session leader of the batch job.

75821 An implementation may indicate to the batch job owner that the batch job has been rerun.  
75822 Whether and how the batch job owner is notified that a batch job is rerun is implementation-  
75823 defined.

75824 A batch server that reruns a batch job may send other implementation-defined signals to the  
75825 session leader of the batch job prior to sending the SIGKILL signal.

75826 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be  
75827 selected for execution before other jobs is implementation-defined.

#### 75828 3.2.3.12 *Select Batch Jobs Request*

75829 A batch client can request from a batch server a list of jobs managed by that server that match a  
75830 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs  
75831 managed by the batch server that receives the request are candidates for selection.

75832 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job  
75833 identifiers that correspond to jobs that meet the selection criteria.

75834 If the batch client is not authorized to query the status of a batch job, the batch server shall not  
75835 select the batch job.

75836 3.2.3.13 *Server Shutdown Request*

75837 A batch server is defined to have shut down when it does not respond to requests from clients  
75838 and does not perform deferred services for jobs. A batch client can request that a batch server  
75839 shut down. Such a request is called a *Server Shutdown Request*.

75840 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut  
75841 down the batch server. The method used to determine whether the user of a client is authorized  
75842 to perform the requested action is implementation-defined.

75843 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons.  
75844 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

75845 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 75846 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the  
75847 batch job and requeue it.
- 75848 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the  
75849 beginning).
- 75850 • Abort the batch job.

75851 3.2.3.14 *Server Status Request*

75852 A batch client can request that a batch server respond with the status and attributes of the batch  
75853 server. Such a request is called a *Server Status Request*.

75854 A batch server shall reject a *Server Status Request* if the following statement is true:

- 75855 • The user of the batch client is not authorized to query the status of the designated server.

75856 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The  
75857 method used to determine whether the user of a client is authorized to perform the requested  
75858 action is implementation-defined.

75859 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch  
75860 client.

75861 3.2.3.15 *Signal Batch Job Request*

75862 A batch client can request that a batch server signal the session leader of a batch job. Such a  
75863 request is called a *Signal Batch Job Request*.

75864 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 75865 • The user of the batch client is not authorized to signal the batch job.
- 75866 • The job is not in the RUNNING state.
- 75867 • The batch server does not manage the designated job.
- 75868 • The requested signal is not supported by the implementation.

75869 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons.  
75870 The method used to determine whether the user of a client is authorized to perform the  
75871 requested action is implementation-defined.

75872 A batch server that accepts a request to signal a batch job shall send the signal requested by the  
75873 batch client to the process group of the session leader of the batch job.

75874 3.2.3.16 *Track Batch Job Request*

75875 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports  
75876 *Track Batch Job Request*, the statements in this section apply and the configuration variable  
75877 POSIX2\_PBS\_TRACK shall be set to 1.

75878 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients  
75879 may use the tracking information to determine the batch server that should receive a batch  
75880 server request.

75881 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a  
75882 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that  
75883 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that  
75884 created the job.

75885 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also  
75886 be sent to other servers as a backup to the primary server. The method by which backup servers  
75887 are specified is implementation-defined.

75888 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,  
75889 then the batch server shall record the current location of the batch job as contained in the  
75890 request.

75891 **3.3 Common Behavior for Batch Environment Utilities**75892 **3.3.1 Batch Job Identifier**

75893 A utility shall recognize *job\_identifiers* of the format:

75894 [sequence\_number] [.server\_name] [@server]

75895 where:

75896 *sequence\_number* An integer that, when combined with *server\_name*, provides a batch job  
75897 identifier that is unique within the batch system.

75898 *server\_name* The name of the batch server to which the batch job was originally submitted.

75899 *server* The name of the batch server that is currently managing the batch job.

75900 If the application omits the batch *server\_name* portion of a batch job identifier, a utility shall use  
75901 the name of a default batch server.

75902 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 75903 • The batch server indicated by *server\_name*, if present
- 75904 • The name of the default batch server
- 75905 • The name of the batch server that is currently managing the batch job

75906 If only @*server* is specified, then the status of all jobs owned by the user on the requested server  
75907 is listed.

75908 The means by which a utility determines the default batch server is implementation-defined.

75909 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility  
75910 shall send the request to the specified server.

75911 A strictly conforming application shall use the syntax described for the job identifier. Whenever  
 75912 a batch job identifier is specified whose syntax is not recognized by an implementation, then a  
 75913 message for each error that occurs shall be written to standard error and the utility shall exit  
 75914 with an exit status greater than zero.

75915 When a batch job identifier is supplied as an argument to a batch utility and the *server\_name*  
 75916 portion of the batch job identifier is omitted, then the utility shall use the name of the default  
 75917 batch server.

75918 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
 75919 portion of the batch job identifier is omitted, then the utility shall use either:

- 75920 • The name of the default batch server
- 75921 or:
- 75922 • The name of the batch server that is currently managing the batch job

75923 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
 75924 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*  
 75925 *Request* to the specified server.

### 75926 3.3.2 Destination

75927 The utility shall recognize a *destination* of the format:

75928 [queue] [@server]

75929 where:

75930 *queue* The name of a valid execution or routing queue at the batch server denoted by  
 75931 @*server*, defined as a string of up to 15 alphanumeric characters in the portable  
 75932 character set (see XBD Section 6.1, on page 125) where the first character is  
 75933 alphabetic.

75934 *server* The name of a batch server, defined as a string of alphanumeric characters in the  
 75935 portable character set.

75936 If the application omits the batch *server* portion of a destination, then the utility shall use either:

- 75937 • The name of the default batch server
- 75938 or:
- 75939 • The name of the batch server that is currently managing the batch job

75940 The means by which a utility determines the default batch server is implementation-defined.

75941 If the application omits the *queue* portion of a destination, then the utility shall use the name of  
 75942 the default queue at the batch server chosen. The means by which a batch server determines its  
 75943 default queue is implementation-defined. If a destination is specified in the *queue@server* form,  
 75944 then the utility shall use the specified queue at the specified server.

75945 A strictly conforming application shall use the syntax described for a destination. Whenever a  
 75946 destination is specified whose syntax is not recognized by an implementation, then a message  
 75947 shall be written to standard error and the utility shall exit with an exit status greater than zero.

75948 **3.3.3 Multiple Keyword-Value Pairs**

75949 For each option that can have multiple keyword-value pair arguments, the following rules shall  
 75950 apply. Examples of options that can have list-oriented option-arguments are `-u value@keyword`  
 75951 and `-l keyword=value`.

75952 1. If a batch utility is presented with a list-oriented option-argument for which a keyword  
 75953 has a corresponding value that begins with a single or double-quote, then the utility shall  
 75954 stop interpreting the input stream for delimiters until a second single or double-quote,  
 75955 respectively, is encountered. This feature allows some flexibility for a <comma> (',' ) or  
 75956 <equals-sign> ('=' ) to be part of the value string for a particular keyword; for example:

75957 `keywd1=' val1, val2' , keywd2="val3, val4"`

75958 **Note:** This may require the user to escape the quotes as in the following command:

75959 `foo -xkeywd1='\ val1, val2\' , keywd2=\"val3, val4\"`

75960 2. If a batch server is presented with a list-oriented attribute that has a keyword that was  
 75961 encountered earlier in the list, then the later entry for that keyword shall replace the  
 75962 earlier entry.

75963 3. If a batch server is presented with a list-oriented attribute that has a keyword without any  
 75964 corresponding value of the form `keyword=` or `@keyword` and the same keyword was  
 75965 encountered earlier in the list, then the prior entry for that keyword shall be ignored by  
 75966 the batch server.

75967 4. If a batch utility is expecting a list-oriented option-argument entry of the form  
 75968 `keyword=value`, but is presented with an entry of the form `keyword` without any  
 75969 corresponding `value`, then the entry shall be treated as though a default value of NULL  
 75970 was assigned (that is, `keyword=NULL`) for entry parsing purposes. The utility shall  
 75971 include only the keyword, not the NULL value, in the associated job attribute.

75972 5. If a batch utility is expecting a list-oriented option-argument entry of the form  
 75973 `value@keyword`, but is presented with an entry of the form `value` without any  
 75974 corresponding `keyword`, then the entry shall be treated as though a keyword of NULL was  
 75975 assigned (that is, `value@NULL`) for entry parsing purposes. The utility shall include only  
 75976 the value, not the NULL keyword, in the associated job attribute.

75977 6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the  
 75978 same keyword, interpreting the keywords, in order, with the last value encountered  
 75979 taking precedence over prior instances of the same keyword. This rule allows, but does  
 75980 not require, a batch utility to preprocess the attribute to remove duplicate keywords.

75981 7. If a batch utility is presented with multiple list-oriented option-arguments on the  
 75982 command line or in script directives, or both, for a single option, then the utility shall  
 75983 concatenate, in order, any command line keyword and value pairs to the end of any  
 75984 directive keyword and value pairs separated by a single <comma> to produce a single  
 75985 string that is an equivalent, valid option-argument. The resulting string shall be assigned  
 75986 to the associated attribute of the batch job (after optionally removing duplicate entries as  
 75987 described in item 6).

(blank page)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

75988

Chapter 4

75989

 Utilities

75990

This chapter contains the definitions of the utilities, as follows:

75991

- Mandatory utilities that are present on every conformant system

75992

- Optional utilities that are present only on systems supporting the associated option; see [Section 1.7.1](#) (on page 7) for information on the options in this volume of POSIX.1-2008

75993

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**admin**

Utilities

75994 **NAME**75995 **admin** — create and administer SCCS files (**DEVELOPMENT**)75996 **SYNOPSIS**

```

75997 xSI admin -i [name] [-n] [-a login] [-d flag] [-e login] [-f flag]
75998         [-m mrlist] [-r rel] [-t name] [-y comment]] newfile
75999
76000 admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
76001         [-t name]] [-y comment]] newfile...
76002
76003 admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t name]] file...
76004
76005 admin -h file...
76006
76007 admin -z file...

```

76004 **DESCRIPTION**

76005 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named  
 76006 file does not exist, it shall be created, and its parameters shall be initialized according to the  
 76007 specified options. Parameters not initialized by an option shall be assigned a default value. If a  
 76008 named file does exist, parameters corresponding to specified options shall be changed, and other  
 76009 parameters shall be left as is.

76010 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files  
 76011 shall be given read-only permission mode. Write permission in the parent directory is required  
 76012 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)  
 76013 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode  
 76014 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file  
 76015 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This  
 76016 ensures that changes are made to the SCCS file only if no errors occur.

76017 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent  
 76018 simultaneous updates to the SCCS file; see *get*.

76019 **OPTIONS**

76020 The *admin* utility shall conform to XBD Section 12.2 (on page 215), except that the *-i*, *-t*, and *-y*  
 76021 options have optional option-arguments. These optional option-arguments shall not be  
 76022 presented as separate arguments. The following options are supported:

76023 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created  
 76024 with control information but without any file data.

76025 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.  
 76026 The text constitutes the first delta of the file (see the *-r* option for the delta  
 76027 numbering scheme). If the *-i* option is used, but the *name* option-argument is  
 76028 omitted, the text shall be obtained by reading the standard input. If this option is  
 76029 omitted, the SCCS file shall be created with control information but without any  
 76030 file data. The *-i* option implies the *-n* option.

76031 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that  
 76032 is, the branch and sequence numbers shall be zero or missing. The level number is  
 76033 optional, and defaults to 1.

76034 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be  
 76035 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

- 76036 • A **-t** option without a *name* option-argument shall cause the removal of  
76037 descriptive text (if any) currently in the SCCS file.
- 76038 • A **-t** option with a *name* option-argument shall cause the text (if any) in the  
76039 named file to replace the descriptive text (if any) currently in the SCCS file.
- 76040 **-f flag** Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.  
76041 Several **-f** options may be supplied on a single *admin* command line.  
76042 Implementations shall recognize the following flags and associated values:
- 76043 **b** Allow use of the **-b** option on a *get* command to create branch deltas.
- 76044 **cceil** Specify the highest release (that is, ceiling), a number less than or equal to  
76045 9999, which may be retrieved by a *get* command for editing. The default  
76046 value for an unspecified **c** flag shall be 9999.
- 76047 **ffloor** Specify the lowest release (that is, floor), a number greater than 0 but less  
76048 than 9999, which may be retrieved by a *get* command for editing. The  
76049 default value for an unspecified **f** flag shall be 1.
- 76050 **dSID** Specify the default delta number (SID) to be used by a *get* command.
- 76051 **istr** Treat the “No ID keywords” message issued by *get* or *delta* as a fatal error.  
76052 In the absence of this flag, the message is only a warning. The message is  
76053 issued if no SCCS identification keywords (see *get*) are found in the text  
76054 retrieved or stored in the SCCS file. If a value is supplied, the application  
76055 shall ensure that the keywords exactly match the given string; however,  
76056 the string shall contain a keyword, and no embedded <newline>  
76057 characters.
- 76058 **j** Allow concurrent *get* commands for editing on the same SID of an SCCS  
76059 file. This allows multiple concurrent updates to the same version of the  
76060 SCCS file.
- 76061 **l~~list~~** Specify a *list* of releases to which deltas can no longer be made (that is, *get*  
76062 **-e** against one of these locked releases fails). Conforming applications  
76063 shall use the following syntax to specify a *list*. Implementations may  
76064 accept additional forms as an extension:
- 76065 <list> ::= a | <range-list>  
76066 <range-list> ::= <range> | <range-list>, <range>  
76067 <range> ::= <SID>
- 76068 The character *a* in the *list* shall be equivalent to specifying all releases for  
76069 the named SCCS file. The non-terminal <SID> in range shall be the delta  
76070 number of an existing delta associated with the SCCS file.
- 76071 **n** Cause *delta* to create a null delta in each of those releases (if any) being  
76072 skipped when a delta is made in a new release (for example, in making  
76073 delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas  
76074 shall serve as anchor points so that branch deltas may later be created  
76075 from them. The absence of this flag shall cause skipped releases to be  
76076 nonexistent in the SCCS file, preventing branch deltas from being created  
76077 from them in the future. During the initial creation of an SCCS file, the **n**  
76078 flag may be ignored; that is, if the **-r** option is used to set the release  
76079 number of the initial SID to a value greater than 1, null deltas need not be  
76080 created for the “skipped” releases.

76081	<b>qtext</b>	Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .
76082		
76083	<b>mmod</b>	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the <b>m</b> flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' .' removed.
76084		
76085		
76086		
76087	<b>ttype</b>	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
76088		
76089	<b>vpgm</b>	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the <b>m</b> option is also used even if its value is null.)
76090		
76091		
76092		
76093		
76094	<b>-d flag</b>	Remove (delete) the specified <i>flag</i> from an SCCS file. Several <b>-d</b> options may be supplied on a single <i>admin</i> command. See the <b>-f</b> option for allowable <i>flag</i> names. (The <b>l</b> list flag gives a <i>list</i> of releases to be unlocked. See the <b>-f</b> option for further description of the <b>l</b> flag and the syntax of a <i>list</i> .)
76095		
76096		
76097		
76098	<b>-a login</b>	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-a</b> options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas.
76099		
76100		
76101		
76102		
76103		
76104		
76105	<b>-e login</b>	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-e</b> options may be used on a single <i>admin</i> command line.
76106		
76107		
76108		
76109	<b>-y[comment]</b>	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the <b>-y</b> option shall result in a default comment line being inserted in the form:
76110		
76111		
76112		"date and time created %s %s by %s", <date>, <time>, <login>
76113		where <date> is expressed in the format of the <i>date</i> utility's %y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
76114		
76115		
76116	<b>-m mrlist</b>	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the <b>v</b> flag is set and the MR numbers are validated if the <b>v</b> flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the <b>v</b> flag is not set or MR validation fails.
76117		
76118		
76119		
76120		
76121	<b>-h</b>	Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
76122		
76123		
76124		

76125        **-z**            Recompute the SCCS file checksum and store it in the first line of the SCCS file (see  
76126                    the **-h** option above). Note that use of this option on a truly corrupted file may  
76127                    prevent future detection of the corruption.

#### 76128 **OPERANDS**

76129        The following operands shall be supported:

76130        *file*            A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*  
76131                    utility shall behave as though each file in the directory were specified as a named  
76132                    file, except that non-SCCS files (last component of the pathname does not begin  
76133                    with **s**.) and unreadable files shall be silently ignored.

76134        *newfile*        A pathname of an SCCS file to be created.

76135        If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each  
76136                    line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-  
76137                    SCCS files and unreadable files shall be silently ignored.

#### 76138 **STDIN**

76139        The standard input shall be a text file used only if **-i** is specified without an option-argument or  
76140                    if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is  
76141                    <SOH> in the POSIX locale, the results are unspecified.

#### 76142 **INPUT FILES**

76143        The existing SCCS files shall be text files of an unspecified format.

76144        The application shall ensure that the file named by the **-i** option's *name* option-argument shall  
76145                    be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results  
76146                    are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the  
76147                    header for this file shall be 99 999 for this delta.

#### 76148 **ENVIRONMENT VARIABLES**

76149        The following environment variables shall affect the execution of *admin*:

76150        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
76151                    (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
76152                    variables used to determine the values of locale categories.)

76153        **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
76154                    internationalization variables.

76155        **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as  
76156                    characters (for example, single-byte as opposed to multi-byte characters in  
76157                    arguments and input files).

76158        **LC\_MESSAGES**

76159                    Determine the locale that should be used to affect the format and contents of  
76160                    diagnostic messages written to standard error and the contents of the default **-y**  
76161                    comment.

76162        **NLSPATH**        Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 76163 **ASYNCHRONOUS EVENTS**

76164        Default.

#### 76165 **STDOUT**

76166        Not used.

76167 **STDERR**

76168 The standard error shall be used only for diagnostic messages.

76169 **OUTPUT FILES**

76170 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a locking *z-file*, as described in *get* (on page 2764), may be created and deleted.

76172 **EXTENDED DESCRIPTION**

76173 None.

76174 **EXIT STATUS**

76175 The following exit values shall be returned:

76176 0 Successful completion.

76177 >0 An error occurred.

76178 **CONSEQUENCES OF ERRORS**

76179 Default.

76180 **APPLICATION USAGE**

76181 It is recommended that directories containing SCCS files be writable by the owner only, and that SCCS files themselves be read-only. The mode of the directories should allow only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

76185 **EXAMPLES**

76186 None.

76187 **RATIONALE**

76188 None.

76189 **FUTURE DIRECTIONS**

76190 None.

76191 **SEE ALSO**

76192 *delta*, *get*, *prs*, *what*

76193 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

76194 **CHANGE HISTORY**

76195 First released in Issue 2.

76196 **Issue 6**

76197 The normative text is reworded to avoid use of the term “must” for application requirements, and to emphasize the term “shall” for implementation requirements.

76199 The grammar is updated.

76200 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

76202 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h** option.

76204 **NAME**

76205 alias — define or display aliases

76206 **SYNOPSIS**76207 alias [*alias-name*[=*string*]...]76208 **DESCRIPTION**

76209 The *alias* utility shall create or redefine alias definitions or write the values of existing alias  
 76210 definitions to standard output. An alias definition provides a string value that shall replace a  
 76211 command name when it is encountered; see [Section 2.3.1](#) (on page 2300).

76212 An alias definition shall affect the current shell execution environment and the execution  
 76213 environments of the subshells of the current shell. When used as specified by this volume of  
 76214 POSIX.1-2008, the alias definition shall not affect the parent process of the current shell nor any  
 76215 utility environment invoked by the shell; see [Section 2.12](#) (on page 2331).

76216 **OPTIONS**

76217 None.

76218 **OPERANDS**

76219 The following operands shall be supported:

76220 *alias-name* Write the alias definition to standard output.76221 *alias-name=string*76222 Assign the value of *string* to the alias *alias-name*.

76223 If no operands are given, all alias definitions shall be written to standard output.

76224 **STDIN**

76225 Not used.

76226 **INPUT FILES**

76227 None.

76228 **ENVIRONMENT VARIABLES**76229 The following environment variables shall affect the execution of *alias*:

76230 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 76231 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization  
 76232 variables used to determine the values of locale categories.)

76233 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 76234 internationalization variables.

76235 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 76236 characters (for example, single-byte as opposed to multi-byte characters in  
 76237 arguments).

76238 *LC\_MESSAGES*

76239 Determine the locale that should be used to affect the format and contents of  
 76240 diagnostic messages written to standard error.

76241 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.76242 **ASYNCHRONOUS EVENTS**

76243 Default.

76244 **STDOUT**

76245 The format for displaying aliases (when no operands or only *name* operands are specified) shall  
76246 be:

76247 `"%s=%s\n", name, value`

76248 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
76249 shell. See the description of shell quoting in Section 2.2 (on page 2298).

76250 **STDERR**

76251 The standard error shall be used only for diagnostic messages.

76252 **OUTPUT FILES**

76253 None.

76254 **EXTENDED DESCRIPTION**

76255 None.

76256 **EXIT STATUS**

76257 The following exit values shall be returned:

76258 0 Successful completion.

76259 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

76260 **CONSEQUENCES OF ERRORS**

76261 Default.

76262 **APPLICATION USAGE**

76263 None.

76264 **EXAMPLES**

76265 1. Create a short alias for a commonly used *ls* command:

76266 `alias lf="ls -CF"`

76267 2. Create a simple "redo" command to repeat previous entries in the command history file:

76268 `alias r='fc -s'`

76269 3. Use 1K units for *du*:

76270 `alias du=du\ -k`

76271 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

76272 `alias nohup="nohup "`

76273 **RATIONALE**

76274 The *alias* description is based on historical KornShell implementations. Known differences exist  
76275 between that and the C shell. The KornShell version was adopted to be consistent with all the  
76276 other KornShell features in this volume of POSIX.1-2008, such as command line editing.

76277 Since *alias* affects the current shell execution environment, it is generally provided as a shell  
76278 regular built-in.

76279 Historical versions of the KornShell have allowed aliases to be exported to scripts that are  
76280 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of  
76281 POSIX.1-2008 only when an explicit extension such as *-x* is used. The standard developers  
76282 considered that aliases were of use primarily to interactive users and that they should normally  
76283 not affect shell scripts called by those users; functions are available to such scripts.

76284 Historical versions of the KornShell had not written aliases in a quoted manner suitable for  
76285 reentry to the shell, but this volume of POSIX.1-2008 has made this a requirement for all similar  
76286 output. Therefore, consistency was chosen over this detail of historical practice.

76287 **FUTURE DIRECTIONS**

76288 None.

76289 **SEE ALSO**

76290 [Section 2.9.5](#) (on page 2324)

76291 XBD [Chapter 8](#) (on page 173)

76292 **CHANGE HISTORY**

76293 First released in Issue 4.

76294 **Issue 6**

76295 This utility is marked as part of the User Portability Utilities option.

76296 The APPLICATION USAGE section is added.

76297 **Issue 7**

76298 The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability  
76299 Utilities is now an option for interactive utilities.

76300 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76301 The first example is changed to remove the creation of an alias for a standard utility that alters  
76302 its behavior to be non-conforming.

## 76303 NAME

76304 ar — create and maintain library archives

## 76305 SYNOPSIS

76306 SD ar -d [-v] archive file...

76307 XSI ar -m [-v] archive file...

76308 ar -m -a [-v] posname archive file...

76309 ar -m -b [-v] posname archive file...

76310 ar -m -i [-v] posname archive file...

76311 XSI ar -p [-v] [-s] archive [file...]

76312 XSI ar -q [-cv] archive file...

76313 ar -r [-cuv] archive file...

76314 XSI ar -r -a [-cuv] posname archive file...

76315 ar -r -b [-cuv] posname archive file...

76316 ar -r -i [-cuv] posname archive file...

76317 XSI ar -t [-v] [-s] archive [file...]

76318 XSI ar -x [-v] [-sCT] archive [file...]

## 76319 DESCRIPTION

76320 The *ar* utility is part of the Software Development Utilities option.

76321 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once  
 76322 an archive has been created, new files can be added, and existing files in an archive can be  
 76323 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the  
 76324 implementation shall format the archive so that it is usable as a library for link editing (see *c99*  
 76325 and *fort77*). When some of the archived files are not valid object files, the suitability of the  
 76326 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire  
 76327 archive shall be printable.

76328 When *ar* creates an archive, it creates administrative information indicating whether a symbol  
 76329 table is present in the archive. When there is at least one object file that *ar* recognizes as such in  
 76330 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is  
 76331 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update  
 76332 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the  
 76333 symbol table to be rebuilt.

76334 All *file* operands can be pathnames. However, files within archives shall be named by a filename,  
 76335 which is the last component of the pathname used when the file was entered into the archive.  
 76336 The comparison of *file* operands to the names of files in archives shall be performed by  
 76337 comparing the last component of the operand to the name of the file in the archive.

76338 It is unspecified whether multiple files in the archive may be identically named. In the case of  
 76339 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive  
 76340 having a name that is the same as the last component of the operand.

76341 **OPTIONS**

76342 The *ar* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

76343 The following options shall be supported:

- 76344 XSI **-a** Position new files in the archive after the file named by the *posname* operand.
- 76345 XSI **-b** Position new files in the archive before the file named by the *posname* operand.
- 76346 **-c** Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.
- 76347
- 76348 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **-T** is also used, to prevent truncated filenames from replacing files with the same prefix.
- 76349
- 76350
- 76351 **-d** Delete one or more *files* from *archive*.
- 76352 XSI **-i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **-b**).
- 76353
- 76354 XSI **-m** Move the named files in the archive. The **-a**, **-b**, or **-i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.
- 76355
- 76356
- 76357 **-p** Write the contents of the *files* in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.
- 76358
- 76359
- 76360 XSI **-q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.
- 76361
- 76362
- 76363 **-r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **-c** option is specified). If no *files* are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the archive shall be appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- 76364
- 76365
- 76366
- 76367
- 76368 XSI
- 76369 XSI **-s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.
- 76370
- 76371
- 76372 **-t** Write a table of contents of *archive* to the standard output. Only the files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.
- 76373
- 76374
- 76375 XSI **-T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.
- 76376
- 76377
- 76378
- 76379 **-u** Update older files in the archive. When used with the **-r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.
- 76380
- 76381

- 76382        **-v**            Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a  
76383 detailed file-by-file description of the archive creation and maintenance activity, as  
76384 described in the STDOUT section.
- 76385                    When used with **-p**, write the name of the file in the archive to the standard output  
76386 before writing the file in the archive itself to the standard output, as described in  
76387 the STDOUT section.
- 76388                    When used with **-t**, include a long listing of information about the files in the  
76389 archive, as described in the STDOUT section.
- 76390        **-x**            Extract the files in the archive named by the *file* operands from *archive*. The  
76391 contents of the archive shall not be changed. If no *file* operands are given, all files  
76392 in the archive shall be extracted. The modification time of each file extracted shall  
76393 be set to the time the file is extracted from the archive.

**OPERANDS**

76394            The following operands shall be supported:

- 76395            *archive*        A pathname of the archive.
- 76396            *file*             A pathname. Only the last component shall be used when comparing against the  
76397 names of files in the archive. If two or more *file* operands have the same last  
76398 pathname component (basename), the results are unspecified. The  
76399 implementation's archive format shall not truncate valid filenames of files added  
76400 to or replaced in the archive.
- 76401
- 76402 XSI        *posname*        The name of a file in the archive, used for relative positioning; see options **-m** and  
76403 **-r**.

**STDIN**

76404            Not used.

**INPUT FILES**

76406            The archive named by *archive* shall be a file in the format created by *ar -r*.

**ENVIRONMENT VARIABLES**

76408            The following environment variables shall affect the execution of *ar*:

- 76410        *LANG*            Provide a default value for the internationalization variables that are unset or null.  
76411 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
76412 variables used to determine the values of locale categories.)
- 76413        *LC\_ALL*         If set to a non-empty string value, override the values of all the other  
76414 internationalization variables.
- 76415        *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
76416 characters (for example, single-byte as opposed to multi-byte characters in  
76417 arguments and input files).
- 76418        *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
76419 diagnostic messages written to standard error.  
76420
- 76421        *LC\_TIME*        Determine the format and content for date and time strings written by *ar -tv*.
- 76422 XSI        *NLSPATH*        Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

76423	<i>TMPDIR</i>	Determine the pathname that overrides the default directory for temporary files, if any.
76424		
76425	<i>TZ</i>	Determine the timezone used to calculate date and time strings written by <i>ar -tv</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
76426		
76427	<b>ASYNCHRONOUS EVENTS</b>	
76428	Default.	
76429	<b>STDOUT</b>	
76430		If the <b>-d</b> option is used with the <b>-v</b> option, the standard output format shall be:
76431		"d - %s\n", <file>
76432		where <i>file</i> is the operand specified on the command line.
76433		If the <b>-p</b> option is used with the <b>-v</b> option, <i>ar</i> shall precede the contents of each file with:
76434		"\n<%s>\n\n", <file>
76435		where <i>file</i> is the operand specified on the command line, if <i>file</i> operands were specified, and the name of the file in the archive if they were not.
76436		
76437		If the <b>-r</b> option is used with the <b>-v</b> option:
76438		• If <i>file</i> is already in the archive, the standard output format shall be:
76439		"r - %s\n", <file>
76440		where < <i>file</i> > is the operand specified on the command line.
76441		• If <i>file</i> is not already in the archive, the standard output format shall be:
76442		"a - %s\n", <file>
76443		where < <i>file</i> > is the operand specified on the command line.
76444		If the <b>-t</b> option is used, <i>ar</i> shall write the names of the files in the archive to the standard output in the format:
76445		
76446		"%s\n", <file>
76447		where <i>file</i> is the operand specified on the command line, if <i>file</i> operands were specified, or the name of the file in the archive if they were not.
76448		
76449		If the <b>-t</b> option is used with the <b>-v</b> option, the standard output format shall be:
76450		"%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>, <group ID>, <number of bytes in member>, <abbreviated month>, <day-of-month>, <hour>, <minute>, <year>, <file>
76451		
76452		
76453		
76454		where:
76455	< <i>file</i> >	Shall be the operand specified on the command line, if <i>file</i> operands were specified, or the name of the file in the archive if they were not.
76456		
76457	< <i>member mode</i> >	
76458		Shall be formatted the same as the < <i>file mode</i> > string defined in the STDOUT section of <i>ls</i> , except that the first character, the < <i>entry type</i> >, is not used; the string represents the file mode of the file in the archive at the time it was added to or replaced in the archive.
76459		
76460		
76461		

76462 The following represent the last-modification time of a file when it was most recently added to  
76463 or replaced in the archive:

76464 <abbreviated month>

76465 Equivalent to the format of the %b conversion specification format in *date*.

76466 <day-of-month>

76467 Equivalent to the format of the %e conversion specification format in *date*.

76468 <hour>

Equivalent to the format of the %H conversion specification format in *date*.

76469 <minute>

Equivalent to the format of the %M conversion specification format in *date*.

76470 <year>

Equivalent to the format of the %Y conversion specification format in *date*.

76471 When *LC\_TIME* does not specify the POSIX locale, a different format and order of presentation  
76472 of these fields relative to each other may be used in a format appropriate in the specified locale.

76473 If the *-x* option is used with the *-v* option, the standard output format shall be:

76474 "x - %s\n", <file>

76475 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
76476 name of the file in the archive if they were not.

#### 76477 **STDERR**

76478 The standard error shall be used only for diagnostic messages. The diagnostic message about  
76479 creating a new archive when *-c* is not specified shall not modify the exit status.

#### 76480 **OUTPUT FILES**

76481 Archives are files with unspecified formats.

#### 76482 **EXTENDED DESCRIPTION**

76483 None.

#### 76484 **EXIT STATUS**

76485 The following exit values shall be returned:

76486 0 Successful completion.

76487 >0 An error occurred.

#### 76488 **CONSEQUENCES OF ERRORS**

76489 Default.

#### 76490 **APPLICATION USAGE**

76491 None.

#### 76492 **EXAMPLES**

76493 None.

#### 76494 **RATIONALE**

76495 The archive format is not described. It is recognized that there are several known *ar* formats,  
76496 which are not compatible. The *ar* utility is included, however, to allow creation of archives that  
76497 are intended for use only on one machine. The archive is specified as a file, and it can be moved  
76498 as a file. This does allow an archive to be moved from one machine to another machine that uses  
76499 the same implementation of *ar*.

76500 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable "archives". This is a not  
76501 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the  
76502 compilers, based on a historical model.

- 76503 In historical implementations, the `-q` option (available on XSI-conforming systems) is known to  
 76504 execute quickly because *ar* does not check on whether the added members are already in the  
 76505 archive. This is useful to bypass the searching otherwise done when creating a large archive  
 76506 piece-by-piece. These remarks may but need not remain true for a brand new implementation of  
 76507 this utility; hence, these remarks have been moved into the RATIONALE.
- 76508 BSD implementations historically required applications to provide the `-s` option whenever the  
 76509 archive was supposed to contain a symbol table. As in this volume of POSIX.1-2008, System V  
 76510 historically creates or updates an archive symbol table whenever an object file is removed from,  
 76511 added to, or updated in the archive.
- 76512 The OPERANDS section requires what might seem to be true without specifying it: the archive  
 76513 cannot truncate the filenames below {NAME\_MAX}. Some historical implementations do so,  
 76514 however, causing unexpected results for the application. Therefore, this volume of POSIX.1-2008  
 76515 makes the requirement explicit to avoid misunderstandings.
- 76516 According to the System V documentation, the options `-dmpqrtx` are not required to begin with  
 76517 a <hyphen> ('-'). This volume of POSIX.1-2008 requires that a conforming application use the  
 76518 leading <hyphen>.
- 76519 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as  
 76520 an example:
- 76521 A file created by *ar* begins with the "magic" string "`!<arch>\n". The rest of the archive  
 76522 is made up of objects, each of which is composed of a header for a file, a possible filename,  
 76523 and the file contents. The header is portable between machine architectures, and, if the file  
 76524 contents are printable, the archive is itself printable.`
- 76525 The header is made up of six ASCII fields, followed by a two-character trailer. The fields  
 76526 are the object name (16 characters), the file last modification time (12 characters), the user  
 76527 and group IDs (each 6 characters), the file mode (8 characters), and the file size (10  
 76528 characters). All numeric fields are in decimal, except for the file mode, which is in octal.
- 76529 The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid`  
 76530 and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field.  
 76531 The two-byte trailer is the string "`\code><newline>`".
- 76532 Only the name field has any provision for overflow. If any filename is more than 16  
 76533 characters in length or contains an embedded space, the string "`#1/`" followed by the  
 76534 ASCII length of the name is written in the name field. The file size (stored in the archive  
 76535 header) is incremented by the length of the name. The name is then written immediately  
 76536 following the archive header.
- 76537 Any unused characters in any of these fields are written as <space> characters. If any fields  
 76538 are their particular maximum number of characters in length, there is no separation  
 76539 between the fields.
- 76540 Objects in the archive are always an even number of bytes long; files that are an odd  
 76541 number of bytes long are padded with a <newline>, although the size in the header does  
 76542 not reflect this.
- 76543 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an  
 76544 object code library, which the linkage editor can use to extract object modules. If the linkage  
 76545 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,  
 76546 *ar* does not require a symbol table.
- 76547 The BSD `-o` option was omitted. It is a rare conforming application that uses *ar* to extract object

76548 code from a library with concern for its modification time, since this can only be of importance  
76549 to *make*. Hence, since this functionality is not deemed important for applications portability, the  
76550 modification time of the extracted files is set to the current time.

76551 There is at least one known implementation (for a small computer) that can accommodate only  
76552 object files for that system, disallowing mixed object and other files. The ability to handle any  
76553 type of file is not only historical practice for most implementations, but is also a reasonable  
76554 expectation.

76555 Consideration was given to changing the output format of *ar -tv* to the same format as the  
76556 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was  
76557 rejected in part because the current *ar* format is commonly used and changes would break  
76558 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a  
76559 <slash>. Changing this to be the user name and group name would not be correct if the archive  
76560 were moved to a machine that contained a different user database. Since *ar* cannot know  
76561 whether the archive was generated on the same machine, it cannot tell what to report.

76562 The text on the *-ur* option combination is historical practice—since one filename can easily  
76563 represent two different files (for example, */a/foo* and */b/foo*), it is reasonable to replace the file in  
76564 the archive even when the modification time in the archive is identical to that in the file system.

#### 76565 FUTURE DIRECTIONS

76566 None.

#### 76567 SEE ALSO

76568 *c99*, *date*, *fort77*, *pax*, *strip*

76569 XBD Chapter 8 (on page 173), Section 12.2 (on page 215), <*unistd.h*>, description of  
76570 {POSIX\_NO\_TRUNC}

#### 76571 CHANGE HISTORY

76572 First released in Issue 2.

#### 76573 Issue 5

76574 The FUTURE DIRECTIONS section is added.

#### 76575 Issue 6

76576 This utility is marked as part of the Software Development Utilities option.

76577 The STDOUT description is changed for the *-v* option to align with the IEEE P1003.2b draft  
76578 standard.

76579 The normative text is reworded to avoid use of the term “must” for application requirements.

76580 The TZ entry is added to the ENVIRONMENT VARIABLES section.

76581 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use  
76582 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being  
76583 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is  
76584 contained in the archive.

76585 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the  
76586 SYNOPSIS. The change was needed since the *-a*, *-b*, and *-i* options are mutually-exclusive, and  
76587 *posname* is required if any of these options is specified.

76588 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description  
76589 of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a  
76590 backquote followed by a <newline>.

76591 **Issue 7**

76592 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
76593 apply.

76594 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76595 The description of the `-t` option is changed to say "Only the files specified ...".

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

76596 **NAME**

76597           asa — interpret carriage-control characters

76598 **SYNOPSIS**76599 FR       asa [*file...*]76600 **DESCRIPTION**76601           The *asa* utility shall write its input files to standard output, mapping carriage-control characters  
76602           from the text files to line-printer control sequences in an implementation-defined manner.76603           The first character of every line shall be removed from the input, and the following actions are  
76604           performed.

76605           If the character removed is:

76606           &lt;space&gt;       The rest of the line is output without change.

76607           0            A &lt;newline&gt; is output, then the rest of the input line.

76608           1            One or more implementation-defined characters that causes an advance to the next  
76609           page shall be output, followed by the rest of the input line.76610           +            The <newline> of the previous line shall be replaced with one or more  
76611           implementation-defined characters that causes printing to return to column  
76612           position 1, followed by the rest of the input line. If the '+' is the first character in  
76613           the input, it shall be equivalent to <space>.76614           The action of the *asa* utility is unspecified upon encountering any character other than those  
76615           listed above as the first character in a line.76616 **OPTIONS**

76617           None.

76618 **OPERANDS**76619           *file*           A pathname of a text file used for input. If no *file* operands are specified, the  
76620           standard input shall be used.76621 **STDIN**76622           The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
76623           operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
76624           the standard input shall not be used. See the INPUT FILES section.76625 **INPUT FILES**

76626           The input files shall be text files.

76627 **ENVIRONMENT VARIABLES**76628           The following environment variables shall affect the execution of *asa*:76629           LANG           Provide a default value for the internationalization variables that are unset or null.  
76630           (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
76631           variables used to determine the values of locale categories.)76632           LC\_ALL          If set to a non-empty string value, override the values of all the other  
76633           internationalization variables.76634           LC\_CTYPE       Determine the locale for the interpretation of sequences of bytes of text data as  
76635           characters (for example, single-byte as opposed to multi-byte characters in  
76636           arguments and input files).

76637	<i>LC_MESSAGES</i>	
76638		Determine the locale that should be used to affect the format and contents of
76639		diagnostic messages written to standard error.
76640	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
76641	<b>ASYNCHRONOUS EVENTS</b>	
76642		Default.
76643	<b>STDOUT</b>	
76644		The standard output shall be the text from the input file modified as described in the
76645		DESCRIPTION section.
76646	<b>STDERR</b>	
76647		None.
76648	<b>OUTPUT FILES</b>	
76649		None.
76650	<b>EXTENDED DESCRIPTION</b>	
76651		None.
76652	<b>EXIT STATUS</b>	
76653		The following exit values shall be returned:
76654	0	All input files were output successfully.
76655	>0	An error occurred.
76656	<b>CONSEQUENCES OF ERRORS</b>	
76657		Default.
76658	<b>APPLICATION USAGE</b>	
76659		None.
76660	<b>EXAMPLES</b>	
76661	1.	The following command:
76662		<i>asa file</i>
76663		permits the viewing of <i>file</i> (created by a program using FORTRAN-style carriage-control
76664		characters) on a terminal.
76665	2.	The following command:
76666		<i>a.out   asa   lp</i>
76667		formats the FORTRAN output of <b>a.out</b> and directs it to the printer.
76668	<b>RATIONALE</b>	
76669		The <i>asa</i> utility is needed to map “standard” FORTRAN 77 output into a form acceptable to
76670		contemporary printers. Usually, <i>asa</i> is used to pipe data to the <i>lp</i> utility; see <i>lp</i> .
76671		This utility is generally used only by FORTRAN programs. The standard developers decided to
76672		retain <i>asa</i> to avoid breaking the historical large base of FORTRAN applications that put carriage-
76673		control characters in their output files. There is no requirement that a system have a FORTRAN
76674		compiler in order to run applications that need <i>asa</i> .
76675		Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII
76676		<carriage-return> in response to a '+' . It is suggested that implementations treat characters
76677		other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.

- 76678            However, the action is listed here as “unspecified”, permitting an implementation to provide  
76679            extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.
- 76680    **FUTURE DIRECTIONS**
- 76681            None.
- 76682    **SEE ALSO**
- 76683            *fort77, lp*
- 76684            XBD [Chapter 8](#) (on page 173)
- 76685    **CHANGE HISTORY**
- 76686            First released in Issue 4.
- 76687    **Issue 6**
- 76688            This utility is marked as part of the FORTRAN Runtime Utilities option.
- 76689            The normative text is reworded to avoid use of the term “must” for application requirements.
- 76690    **Issue 7**
- 76691            Austin Group Interpretation 1003.1-2001 #092 is applied.
- 76692            SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

76693 **NAME**

76694 at — execute commands at a later time

76695 **SYNOPSIS**76696 at [-m] [-f *file*] [-q *queuename*] -t *time\_arg*76697 at [-m] [-f *file*] [-q *queuename*] *timespec...*76698 at -r *at\_job\_id...*76699 at -l -q *queuename*76700 at -l [*at\_job\_id...*]76701 **DESCRIPTION**76702 The *at* utility shall read commands from standard input and group them together as an *at-job*, to  
76703 be executed at a later time.76704 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process  
76705 group with no controlling terminal, except that the environment variables, current working  
76706 directory, file creation mask, and other implementation-defined execution-time attributes in  
76707 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.76708 When the *at-job* is submitted, the *at\_job\_id* and scheduled time shall be written to standard error.  
76709 The *at\_job\_id* is an identifier that shall be a string consisting solely of alphanumeric characters  
76710 and the <period> character. The *at\_job\_id* shall be assigned by the system when the job is  
76711 scheduled such that it uniquely identifies a particular job.76712 User notification and the processing of the job's standard output and standard error are  
76713 described under the **-m** option.76714 XSI Users shall be permitted to use *at* if their name appears in the file **at.allow** which is located in an  
76715 implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in  
76716 an implementation-defined directory, shall be checked to determine whether the user shall be  
76717 denied access to *at*. If neither file exists, only a process with appropriate privileges shall be  
76718 allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The  
76719 **at.allow** and **at.deny** files shall consist of one user name per line.76720 **OPTIONS**76721 The *at* utility shall conform to XBD [Section 12.2](#) (on page 215).

76722 The following options shall be supported:

76723 **-f file** Specify the pathname of a file to be used as the source of the *at-job*, instead of  
76724 standard input.76725 **-l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at\_job\_id*  
76726 operands are specified. If *at\_job\_ids* are specified, report only information for these  
76727 jobs. The output shall be written to standard output.76728 **-m** Send mail to the invoking user after the *at-job* has run, announcing its completion.  
76729 Standard output and standard error produced by the *at-job* shall be mailed to the  
76730 user as well, unless redirected elsewhere. Mail shall be sent even if the job  
76731 produces no output.76732 If **-m** is not used, the job's standard output and standard error shall be provided to  
76733 the user by means of mail, unless they are redirected elsewhere; if there is no such  
76734 output to provide, the implementation need not notify the user of the job's  
76735 completion.

- 76736            **-q** *queuename*  
 76737            Specify in which queue to schedule a job for submission. When used with the **-l**  
 76738            option, limit the search to that particular queue. By default, at-jobs shall be  
 76739            scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see  
 76740            *batch*. The meanings of all other *queuenames* are implementation-defined. If **-q** is  
 76741            specified along with either of the **-t** *time\_arg* or *timespec* arguments, the results are  
 76742            unspecified.
- 76743            **-r**            Remove the jobs with the specified *at\_job\_id* operands that were previously  
 76744            scheduled by the *at* utility.
- 76745            **-t** *time\_arg*    Submit the job to be run at the time specified by the *time* option-argument, which  
 76746            the application shall ensure has the format as specified by the *touch -t time* utility.

## 76747 OPERANDS

76748 The following operands shall be supported:

- 76749            *at\_job\_id*      The name reported by a previous invocation of the *at* utility at the time the job was  
 76750            scheduled.
- 76751            *timespec*      Submit the job to be run at the date and time specified. All of the *timespec* operands  
 76752            are interpreted as if they were separated by <space> characters and concatenated,  
 76753            and shall be parsed as described in the grammar at the end of this section. The date  
 76754            and time shall be interpreted as being in the timezone of the user (as determined  
 76755            by the *TZ* variable), unless a timezone name appears as part of *time*, below.
- 76756            In the POSIX locale, the following describes the three parts of the time specification  
 76757            string. All of the values from the *LC\_TIME* categories in the POSIX locale shall be  
 76758            recognized in a case-insensitive manner.
- 76759            *time*            The time can be specified as one, two, or four digits. One-digit and  
 76760            two-digit numbers shall be taken to be hours; four-digit numbers to  
 76761            be hours and minutes. The time can alternatively be specified as two  
 76762            numbers separated by a <colon>, meaning *hour:minute*. An AM/PM  
 76763            indication (one of the values from the **am\_pm** keywords in the  
 76764            *LC\_TIME* locale category) can follow the time; otherwise, a 24-hour  
 76765            clock time shall be understood. A timezone name can also follow to  
 76766            further qualify the time. The acceptable timezone names are  
 76767            implementation-defined, except that they shall be case-insensitive  
 76768            and the string **utc** is supported to indicate the time is in Coordinated  
 76769            Universal Time. In the POSIX locale, the *time* field can also be one of  
 76770            the following tokens:
- 76771                       **midnight**      Indicates the time 12:00 am (00:00).
- 76772                       **noon**            Indicates the time 12:00 pm.
- 76773                       **now**            Indicates the current day and time. Invoking *at* <**now**>  
 76774            shall submit an at-job for potentially immediate  
 76775            execution (that is, subject only to unspecified  
 76776            scheduling delays).
- 76777            *date*            An optional *date* can be specified as either a month name (one of the  
 76778            values from the **mon** or **abmon** keywords in the *LC\_TIME* locale  
 76779            category) followed by a day number (and possibly year number  
 76780            preceded by a comma), or a day of the week (one of the values from  
 76781            the **day** or **abday** keywords in the *LC\_TIME* locale category). In the

76782 POSIX locale, two special days shall be recognized:

76783 **today** Indicates the current day.

76784 **tomorrow** Indicates the day following the current day.

76785 If no *date* is given, **today** shall be assumed if the given time is greater  
76786 than the current time, and **tomorrow** shall be assumed if it is less. If  
76787 the given month is less than the current month (and no year is given),  
76788 next year shall be assumed.

76789 *increment* The optional *increment* shall be a number preceded by a <plus-sign>  
76790 ('+') and suffixed by one of the following: **minutes**, **hours**, **days**,  
76791 **weeks**, **months**, or **years**. (The singular forms shall also be accepted.)  
76792 The keyword **next** shall be equivalent to an increment number of +1.  
76793 For example, the following are equivalent commands:

76794 at 2pm + 1 week  
76795 at 2pm next week

76796 The following grammar describes the precise format of *timespec* in the POSIX locale. The general  
76797 conventions for this style of grammar are described in Section 1.3 (on page 2287). This formal  
76798 syntax shall take precedence over the preceding text syntax description. The longest possible  
76799 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space  
76800 shall also delimit tokens.

```
76801 %token hr24clock_hr_min
76802 %token hr24clock_hour
76803 /*
76804 An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
76805 or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
76806 may be any of the single digits [0,9], or may be double digits, ranging
76807 from [00,23]. If an hr24clock_hr_min is a four-digit number, the
76808 first two digits shall be a valid hr24clock_hour, while the last two
76809 represent the number of minutes, from [00,59].
76810 */
```

```
76811 %token wallclock_hr_min
76812 %token wallclock_hour
76813 /*
76814 A wallclock_hr_min is a one, two-digit, or four-digit number.
76815 A one digit or two-digit number constitutes a wallclock_hour.
76816 A wallclock_hour may be any of the single digits [1,9], or may
76817 be double digits, ranging from [01,12]. If a wallclock_hr_min
76818 is a four-digit number, the first two digits shall be a valid
76819 wallclock_hour, while the last two represent the number of
76820 minutes, from [00,59].
76821 */
```

```
76822 %token minute
76823 /*
76824 A minute is a one or two-digit number whose value can be [0,9]
76825 or [00,59].
76826 */
```

```
76827 %token day_number
76828 /*
```

```

76829     A day_number is a number in the range appropriate for the particular
76830     month and year specified by month_name and year_number, respectively.
76831     If no year_number is given, the current year is assumed if the given
76832     date and time are later this year. If no year_number is given and
76833     the date and time have already occurred this year and the month is
76834     not the current month, next year is the assumed year.
76835     */

76836     %token year_number
76837     /*
76838     A year_number is a four-digit number representing the year A.D., in
76839     which the at_job is to be run.
76840     */

76841     %token inc_number
76842     /*
76843     The inc_number is the number of times the succeeding increment
76844     period is to be added to the specified date and time.
76845     */

76846     %token timezone_name
76847     /*
76848     The name of an optional timezone suffix to the time field, in an
76849     implementation-defined format.
76850     */

76851     %token month_name
76852     /*
76853     One of the values from the mon or abmon keywords in the LC_TIME
76854     locale category.
76855     */

76856     %token day_of_week
76857     /*
76858     One of the values from the day or abday keywords in the LC_TIME
76859     locale category.
76860     */

76861     %token am_pm
76862     /*
76863     One of the values from the am_pm keyword in the LC_TIME locale
76864     category.
76865     */

76866     %start timespec
76867     %%
76868     timespec      : time
76869                   | time date
76870                   | time increment
76871                   | time date increment
76872                   | nowspec
76873                   ;

76874     nowspec      : "now"
76875                   | "now" increment
76876                   ;

```

```

76877     time           : hr24clock_hr_min
76878     | hr24clock_hr_min timezone_name
76879     | hr24clock_hour ":" minute
76880     | hr24clock_hour ":" minute timezone_name
76881     | wallclock_hr_min am_pm
76882     | wallclock_hr_min am_pm timezone_name
76883     | wallclock_hour ":" minute am_pm
76884     | wallclock_hour ":" minute am_pm timezone_name
76885     | "noon"
76886     | "midnight"
76887     ;

76888     date            : month_name day_number
76889     | month_name day_number "," year_number
76890     | day_of_week
76891     | "today"
76892     | "tomorrow"
76893     ;

76894     increment       : "+" inc_number inc_period
76895     | "next" inc_period
76896     ;

76897     inc_period       : "minute" | "minutes"
76898     | "hour" | "hours"
76899     | "day" | "days"
76900     | "week" | "weeks"
76901     | "month" | "months"
76902     | "year" | "years"
76903     ;

```

**76904 STDIN**

76905 The standard input shall be a text file consisting of commands acceptable to the shell command  
76906 language described in [Chapter 2](#) (on page 2297). The standard input shall only be used if no `-f`  
76907 `file` option is specified.

**76908 INPUT FILES**

76909 See the STDIN section.

76910 XSI The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory,  
76911 shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
76912 denied access to the `at` and `batch` utilities.

**76913 ENVIRONMENT VARIABLES**

76914 The following environment variables shall affect the execution of `at`:

76915 `LANG` Provide a default value for the internationalization variables that are unset or null.  
76916 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
76917 variables used to determine the values of locale categories.)

76918 `LC_ALL` If set to a non-empty string value, override the values of all the other  
76919 internationalization variables.

76920 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
76921 characters (for example, single-byte as opposed to multi-byte characters in  
76922 arguments and input files).

76923	<i>LC_MESSAGES</i>	
76924		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
76925		
76926		
76927	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
76928	<i>LC_TIME</i>	Determine the format and contents for date and time strings written and accepted by <i>at</i> .
76929		
76930	<i>SHELL</i>	Determine a name of a command interpreter to be used to invoke the <i>at</i> -job: If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; or any of the preceding accompanied by a warning diagnostic about which was chosen.
76931		
76932		
76933		
76934		
76935	<i>TZ</i>	Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it shall override <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
76936		
76937		
76938		
76939		
76940	<b>ASYNCHRONOUS EVENTS</b>	
76941		Default.
76942	<b>STDOUT</b>	
76943		When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.
76944		
76945		In the POSIX locale, the following shall be written to the standard output for each job when jobs are listed in response to the <i>-l</i> option:
76946		
76947		<code>"%s\t%s\n", at_job_id, &lt;date&gt;</code>
76948		where <i>date</i> shall be equivalent in format to the output of:
76949		<code>date +"%a %b %e %T %Y"</code>
76950		The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).
76951		
76952	<b>STDERR</b>	
76953		In the POSIX locale, the following shall be written to standard error when a job has been successfully submitted:
76954		
76955		<code>"job %s at %s\n", at_job_id, &lt;date&gt;</code>
76956		where <i>date</i> has the same format as that described in the STDOUT section. Neither this, nor warning messages concerning the selection of the command interpreter, shall be considered a diagnostic that changes the exit status.
76957		
76958		
76959		Diagnostic messages, if any, shall be written to standard error.
76960	<b>OUTPUT FILES</b>	
76961		None.
76962	<b>EXTENDED DESCRIPTION</b>	
76963		None.

76964 **EXIT STATUS**

76965 The following exit values shall be returned:

76966 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

76967 &gt;0 An error occurred.

76968 **CONSEQUENCES OF ERRORS**

76969 The job shall not be scheduled, removed, or listed.

76970 **APPLICATION USAGE**76971 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other  
76972 cultures may be supported with substantially different interfaces, although implementations are  
76973 encouraged to provide comparable levels of functionality.76974 Since the commands run in a separate shell invocation, running in a separate process group with  
76975 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking  
76976 environment are lost.76977 Some implementations do not allow substitution of different shells using *SHELL*. System V  
76978 systems, for example, have used the login shell value for the user in */etc/passwd*. To select  
76979 reliably another command interpreter, the user must include it as part of the script, such as:76980 \$ at 1800  
76981 myshell myscript  
76982 EOT  
76983 **job ... at ...**  
76984 \$76985 **EXAMPLES**

76986 1. This sequence can be used at a terminal:

76987 at -m 0730 tomorrow  
76988 sort < file >outfile  
76989 EOT76990 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
76991 command procedure (the sequence of output redirection specifications is significant):76992 at now + 1 hour <<!  
76993 diff file1 file2 2>&1 >outfile | mailx mygroup  
76994 !76995 3. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this  
76996 daily processing script named **my.daily** runs every day (although *crontab* is a more  
76997 appropriate vehicle for such work):76998 # my.daily runs every day  
76999 *daily processing*  
77000 at now tomorrow < my.daily77001 4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as  
77002 there are no ambiguities. Examples of various times and operand presentation include:77003 at 0815am Jan 24  
77004 at 8 :15amjan24  
77005 at now "+ 1day"  
77006 at 5 pm FRIday  
77007 at '17

77008                   utc+  
77009                   30minutes'

#### 77010 RATIONALE

77011                   The *at* utility reads from standard input the commands to be executed at a later time. It may be  
77012                   useful to redirect standard output and standard error within the specified commands.

77013                   The *-t time* option was added as a new capability to support an internationalized way of  
77014                   specifying a time for execution of the submitted job.

77015                   Early proposals added a “jobname” concept as a way of giving submitted jobs names that are  
77016                   meaningful to the user submitting them. The historical, system-specified *at\_job\_id* gives no  
77017                   indication of what the job is. Upon further reflection, it was decided that the benefit of this was  
77018                   not worth the change in historical interface. The *at* functionality is useful in simple  
77019                   environments, but in large or complex situations, the functionality provided by the Batch  
77020                   Services option is more suitable.

77021                   The *-q* option historically has been an undocumented option, used mainly by the *batch* utility.

77022                   The System V *-m* option was added to provide a method for informing users that an at-job had  
77023                   completed. Otherwise, users are only informed when output to standard error or standard  
77024                   output are not redirected.

77025                   The behavior of *at <now>* was changed in an early proposal from being unspecified to  
77026                   submitting a job for potentially immediate execution. Historical BSD *at* implementations support  
77027                   this. Historical System V implementations give an error in that case, but a change to the System  
77028                   V versions should have no backwards-compatibility ramifications.

77029                   On BSD-based systems, a *-u user* option has allowed those with appropriate privileges to access  
77030                   the work of other users. Since this is primarily a system administration feature and is not  
77031                   universally implemented, it has been omitted. Similarly, a specification for the output format for  
77032                   a user with appropriate privileges viewing the queues of other users has been omitted.

77033                   The *-f file* option from System V is used instead of the BSD method of using the last operand as  
77034                   the pathname. The BSD method is ambiguous—does:

77035                   at 1200 friday

77036                   mean the same thing if there is a file named **friday** in the current directory?

77037                   The *at\_job\_id* is composed of a limited character set in historical practice, and it is mandated here  
77038                   to invalidate systems that might try using characters that require shell quoting or that could not  
77039                   be easily parsed by shell scripts.

77040                   The *at* utility varies between System V and BSD systems in the way timezones are used. On  
77041                   System V systems, the *TZ* variable affects the at-job submission times and the times displayed  
77042                   for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved  
77043                   with the current specification. If the user wishes to have the timezone default to that of the  
77044                   system, they merely need to issue the *at* command immediately following an unsetting or null  
77045                   assignment to *TZ*. For example:

77046                   TZ= at noon ...

77047                   gives the desired BSD result.

77048                   While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with  
77049                   respect to the digit strings, a lexical analyzer would probably be written to look for and return  
77050                   digit strings in those cases. The parser could then check whether the digit string returned is a  
77051                   valid *day\_number*, *year\_number*, and so on, based on the context.

77052 **FUTURE DIRECTIONS**

77053 None.

77054 **SEE ALSO**77055 *batch, crontab*77056 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)77057 **CHANGE HISTORY**

77058 First released in Issue 2.

77059 **Issue 6**

77060 This utility is marked as part of the User Portability Utilities option.

77061 The following new requirements on POSIX implementations derive from alignment with the  
77062 Single UNIX Specification:

- 77063
- If **-m** is not used, the job's standard output and standard error are provided to the user by  
77064 mail.

77065 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are  
77066 specified.

77067 The normative text is reworded to avoid use of the term "must" for application requirements.

77068 **Issue 7**77069 The *at* utility is moved from the User Portability Utilities option to the Base. User Portability  
77070 Utilities is now an option for interactive utilities.77071 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
77072 by the *at* utility.

77073 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**awk**

Utilities

77074 **NAME**

77075 awk — pattern scanning and processing language

77076 **SYNOPSIS**77077 awk [-F *ERE*] [-v *assignment*]... *program* [*argument*...]77078 awk [-F *ERE*] -f *progfile* [-f *progfile*]... [-v *assignment*]...  
77079 [*argument*...]77080 **DESCRIPTION**77081 The *awk* utility shall execute programs written in the *awk* programming language, which is  
77082 specialized for textual data manipulation. An *awk* program is a sequence of patterns and  
77083 corresponding actions. When input is read that matches a pattern, the action associated with that  
77084 pattern is carried out.77085 Input shall be interpreted as a sequence of records. By default, a record is a line, less its  
77086 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of  
77087 input shall be matched in turn against each pattern in the program. For each pattern matched,  
77088 the associated action shall be executed.77089 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field  
77090 is a string of non-<blank> characters. This default white-space field delimiter can be changed by  
77091 using the **FS** built-in variable or **-F *ERE***. The *awk* utility shall denote the first field in a record \$1,  
77092 the second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any other field  
77093 causes the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields and the **NF**  
77094 built-in variable.77095 **OPTIONS**77096 The *awk* utility shall conform to XBD Section 12.2 (on page 215).

77097 The following options shall be supported:

77098 **-F *ERE*** Define the input field separator to be the extended regular expression *ERE*, before  
77099 any input is read; see [Regular Expressions](#) (on page 2439).77100 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. If multiple  
77101 instances of this option are specified, the concatenation of the files specified as  
77102 *progfile* in the order specified shall be the *awk* program. The *awk* program can  
77103 alternatively be specified in the command line as a single argument.77104 **-v *assignment***77105 The application shall ensure that the *assignment* argument is in the same form as an  
77106 *assignment* operand. The specified variable assignment shall occur prior to  
77107 executing the *awk* program, including the actions associated with **BEGIN** patterns  
77108 (if any). Multiple occurrences of this option can be specified.77109 **OPERANDS**

77110 The following operands shall be supported:

77111 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*  
77112 program. The application shall supply the *program* operand as a single argument to  
77113 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.77114 *argument* Either of the following two types of *argument* can be intermixed:77115 *file* A pathname of a file that contains the input to be read, which is  
77116 matched against the set of patterns in the program. If no *file* operands  
77117 are specified, or if a *file* operand is '-', the standard input shall be  
77118 used.

77119 *assignment* An operand that begins with an <underscore> or alphabetic  
 77120 character from the portable character set (see the table in XBD Section  
 77121 6.1, on page 125), followed by a sequence of underscores, digits, and  
 77122 alphabetic characters from the portable character set, followed by the '='  
 77123 character, shall specify a variable assignment rather than a pathname.  
 77124 The characters before the '=' represent the name of an *awk* variable;  
 77125 if that name is an *awk* reserved word (see Grammar, on page 2447)  
 77126 the behavior is undefined. The characters following the <equals-  
 77127 sign> shall be interpreted as if they appeared in the *awk* program  
 77128 preceded and followed by a double-quote ('"') character, as a  
 77129 **STRING** token (see Grammar, on page 2447), except that if the last  
 77130 character is an unescaped <backslash>, it shall be interpreted as a  
 77131 literal <backslash> rather than as the first character of the sequence  
 77132 "\". The variable shall be assigned the value of that **STRING**  
 77133 token and, if appropriate, shall be considered a *numeric string* (see  
 77134 Expressions in *awk*, on page 2433), the variable shall also be assigned  
 77135 its numeric value. Each such variable assignment shall occur just  
 77136 prior to the processing of the following *file*, if any. Thus, an  
 77137 assignment before the first *file* argument shall be executed after the  
 77138 **BEGIN** actions (if any), while an assignment after the last *file*  
 77139 argument shall occur before the **END** actions (if any). If there are no  
 77140 *file* arguments, assignments shall be executed before processing the  
 77141 standard input.

#### 77142 **STDIN**

77143 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-' ;  
 77144 see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is  
 77145 otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk*  
 77146 shall exit with a return status of zero.

#### 77147 **INPUT FILES**

77148 Input files to the *awk* program from any of the following sources shall be text files:

- 77149 • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV**  
 77150 and **ARGC**
- 77151 • Standard input in the absence of any *file* operands
- 77152 • Arguments to the **getline** function

77153 Whether the variable **RS** is set to a value other than a <newline> or not, for these files,  
 77154 implementations shall support records terminated with the specified separator up to  
 77155 {LINE\_MAX} bytes and may support longer records.

77156 If **-f *progfile*** is specified, the application shall ensure that the files named by each of the *progfile*  
 77157 option-arguments are text files and their concatenation, in the same order as they appear in the  
 77158 arguments, is an *awk* program.

#### 77159 **ENVIRONMENT VARIABLES**

77160 The following environment variables shall affect the execution of *awk*:

77161 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 77162 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 77163 variables used to determine the values of locale categories.)

77164	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
77165		
77166	<i>LC_COLLATE</i>	
77167		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and in comparisons of string values.
77168		
77169		
77170	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of uppercase and lowercase characters for the <b>toupper</b> and <b>tolower</b> functions.
77171		
77172		
77173		
77174		
77175	<i>LC_MESSAGES</i>	
77176		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
77177		
77178	<i>LC_NUMERIC</i>	
77179		Determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output.
77180		Regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing <i>awk</i> programs (including assignments in command line arguments).
77181		
77182		
77183		
77184	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
77185	<i>PATH</i>	Determine the search path when looking for commands executed by <i>system(expr)</i> , or input and output pipes; see XBD Chapter 8 (on page 173).
77186		
77187		In addition, all environment variables shall be visible via the <i>awk</i> variable <b>ENVIRON</b> .
77188	<b>ASYNCHRONOUS EVENTS</b>	
77189		Default.
77190	<b>STDOUT</b>	
77191		The nature of the output files depends on the <i>awk</i> program.
77192	<b>STDERR</b>	
77193		The standard error shall be used only for diagnostic messages.
77194	<b>OUTPUT FILES</b>	
77195		The nature of the output files depends on the <i>awk</i> program.
77196	<b>EXTENDED DESCRIPTION</b>	
77197		<b>Overall Program Structure</b>
77198		An <i>awk</i> program is composed of pairs of the form:
77199		<i>pattern</i> { <i>action</i> }
77200		Either the pattern or the action (including the enclosing brace characters) can be omitted.
77201		A missing pattern shall match any record of input, and a missing action shall be equivalent to:
77202		{ <i>print</i> }
77203		Execution of the <i>awk</i> program shall start by first executing the actions associated with all <b>BEGIN</b> patterns in the order they occur in the program. Then each <i>file</i> operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record
77204		
77205		

77206 separator is seen (<newline> by default). Before the first reference to a field in the record is  
 77207 evaluated, the record shall be split into fields, according to the rules in [Regular Expressions](#) (on  
 77208 page 2439), using the value of **FS** that was current at the time the record was read. Each pattern  
 77209 in the program then shall be evaluated in the order of occurrence, and the action associated with  
 77210 each pattern that matches the current record executed. The action for a matching pattern shall be  
 77211 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**  
 77212 patterns shall be executed in the order they occur in the program.

### 77213 Expressions in awk

77214 Expressions describe computations used in *patterns* and *actions*. In the following table, valid  
 77215 expression operations are given in groups from highest precedence first to lowest precedence  
 77216 last, with equal-precedence operators grouped between horizontal lines. In expression  
 77217 evaluation, where the grammar is formally ambiguous, higher precedence operators shall be  
 77218 evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent  
 77219 any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side  
 77220 of an assignment operator). The precise syntax of expressions is given in [Grammar](#) (on page  
 77221 2447).

77222 **Table 4-1** Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
( <i>expr</i> )	Grouping	Type of <i>expr</i>	N/A
\$ <i>expr</i>	Field reference	String	N/A
<i>lvalue</i> ++	Post-increment	Numeric	N/A
<i>lvalue</i> --	Post-decrement	Numeric	N/A
++ <i>lvalue</i>	Pre-increment	Numeric	N/A
-- <i>lvalue</i>	Pre-decrement	Numeric	N/A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N/A
+ <i>expr</i>	Unary plus	Numeric	N/A
- <i>expr</i>	Unary minus	Numeric	N/A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left
<i>expr</i> + <i>expr</i>	Addition	Numeric	Left
<i>expr</i> - <i>expr</i>	Subtraction	Numeric	Left
<i>expr</i> <i>expr</i>	String concatenation	String	Left
<i>expr</i> < <i>expr</i>	Less than	Numeric	None
<i>expr</i> <= <i>expr</i>	Less than or equal to	Numeric	None
<i>expr</i> != <i>expr</i>	Not equal to	Numeric	None
<i>expr</i> == <i>expr</i>	Equal to	Numeric	None
<i>expr</i> > <i>expr</i>	Greater than	Numeric	None
<i>expr</i> >= <i>expr</i>	Greater than or equal to	Numeric	None
<i>expr</i> ~ <i>expr</i>	ERE match	Numeric	None
<i>expr</i> !~ <i>expr</i>	ERE non-match	Numeric	None
<i>expr</i> in <i>array</i>	Array membership	Numeric	Left
( <i>index</i> ) in <i>array</i>	Multi-dimension array	Numeric	Left

	Syntax	Name	Type of Result	Associativity
77250		membership		
77251				
77252	<code>expr &amp;&amp; expr</code>	Logical AND	Numeric	Left
77253	<code>expr    expr</code>	Logical OR	Numeric	Left
77254	<code>expr1 ? expr2 : expr3</code>	Conditional expression	Type of selected <i>expr2</i> or <i>expr3</i>	Right
77255				
77256	<code>lvalue ^= expr</code>	Exponentiation assignment	Numeric	Right
77257	<code>lvalue %= expr</code>	Modulus assignment	Numeric	Right
77258	<code>lvalue *= expr</code>	Multiplication assignment	Numeric	Right
77259	<code>lvalue /= expr</code>	Division assignment	Numeric	Right
77260	<code>lvalue += expr</code>	Addition assignment	Numeric	Right
77261	<code>lvalue -= expr</code>	Subtraction assignment	Numeric	Right
77262	<code>lvalue = expr</code>	Assignment	Type of <i>expr</i>	Right

77263 Each expression shall have either a string value, a numeric value, or both. Except as stated for  
 77264 specific contexts, the value of an expression shall be implicitly converted to the type needed for  
 77265 the context in which it is used. A string value shall be converted to a numeric value either by the  
 77266 equivalent of the following calls to functions defined by the ISO C standard:

```
77267 setlocale(LC_NUMERIC, "");
77268 numeric_value = atof(string_value);
```

77269 or by converting the initial portion of the string to type **double** representation as follows:

77270 The input string is decomposed into two parts: an initial, possibly empty, sequence of  
 77271 white-space characters (as specified by `isspace()`) and a subject sequence interpreted as a  
 77272 floating-point constant.

77273 The expected form of the subject sequence is an optional '+' or '-' sign, then a non-  
 77274 empty sequence of digits optionally containing a <period>, then an optional exponent  
 77275 part. An exponent part consists of 'e' or 'E', followed by an optional sign, followed by  
 77276 one or more decimal digits.

77277 The sequence starting with the first digit or the <period> (whichever occurs first) is  
 77278 interpreted as a floating constant of the C language, and if neither an exponent part nor a  
 77279 <period> appears, a <period> is assumed to follow the last digit in the string. If the subject  
 77280 sequence begins with a minus-sign, the value resulting from the conversion is negated.

77281 A numeric value that is exactly equal to the value of an integer (see [Section 1.1.2](#), on page 2283)  
 77282 shall be converted to a string by the equivalent of a call to the **sprintf** function (see [String](#)  
 77283 [Functions](#), on page 2444) with the string "%d" as the *fmt* argument and the numeric value being  
 77284 converted as the first and only *expr* argument. Any other numeric value shall be converted to a  
 77285 string by the equivalent of a call to the **sprintf** function with the value of the variable  
 77286 **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only  
 77287 *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a  
 77288 floating-point format specification. This volume of POSIX.1-2008 specifies no explicit  
 77289 conversions between numbers and strings. An application can force an expression to be treated  
 77290 as a number by adding zero to it, or can force it to be treated as a string by concatenating the null  
 77291 string (" ") to it.

77292 A string value shall be considered a *numeric string* if it comes from one of the following:

- 77293 1. Field variables

- 77294 2. Input from the *getline()* function
- 77295 3. **FILENAME**
- 77296 4. **ARGV** array elements
- 77297 5. **ENVIRON** array elements
- 77298 6. Array elements created by the *split()* function
- 77299 7. A command line variable assignment
- 77300 8. Variable assignment from another numeric string variable

77301 and an implementation-dependent condition corresponding to either case (a) or (b) below is met.

77302

- 77303 a. After the equivalent of the following calls to functions defined by the ISO C standard, *string\_value\_end* would differ from *string\_value*, and any characters before the terminating null character in *string\_value\_end* would be <blank> characters:

```
77306 char *string_value_end;
77307 setlocale(LC_NUMERIC, "");
77308 numeric_value = strtod (string_value, &string_value_end);
```

- 77309 b. After all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in [Grammar](#) (on page 2447):

- 77312 — All leading and trailing <blank> characters are discarded.
- 77313 — If the first non-<blank> is '+' or '-', it is discarded.
- 77314 — Each occurrence of the decimal point character from the current locale is changed to a <period>.

77316 In case (a) the numeric value of the *numeric string* shall be the value that would be returned by the *strtod()* call. In case (b) if the first non-<blank> is '-', the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token; otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

77322 When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean context shall be one of the following:

- 77326 • The first subexpression of a conditional expression
- 77327 • An expression operated on by logical NOT, logical AND, or logical OR
- 77328 • The second expression of a **for** statement
- 77329 • The expression of an **if** statement
- 77330 • The expression of the **while** clause in either a **while** or **do...while** statement
- 77331 • An expression used as a pattern (as in Overall Program Structure)

77332 All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C standard (see [Section 1.1.2](#), on page 2283).

77334 The value of the expression:

77335 `expr1 ^ expr2`

77336 shall be equivalent to the value returned by the ISO C standard function call:

77337 `pow(expr1, expr2)`

77338 The expression:

77339 `lvalue ^= expr`

77340 shall be equivalent to the ISO C standard expression:

77341 `lvalue = pow(lvalue, expr)`

77342 except that `lvalue` shall be evaluated only once. The value of the expression:

77343 `expr1 % expr2`

77344 shall be equivalent to the value returned by the ISO C standard function call:

77345 `fmod(expr1, expr2)`

77346 The expression:

77347 `lvalue %= expr`

77348 shall be equivalent to the ISO C standard expression:

77349 `lvalue = fmod(lvalue, expr)`

77350 except that `lvalue` shall be evaluated only once.

77351 Variables and fields shall be set by the assignment statement:

77352 `lvalue = expression`

77353 and the type of *expression* shall determine the resulting variable type. The assignment includes the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which shall produce a numeric result. The left-hand side of an assignment and the target of increment and decrement operators can be one of a variable, an array with index, or a field selector.

77357 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not be declared. They shall initially be empty, and their sizes shall change dynamically. The subscripts, or element identifiers, are strings, providing a type of associative array capability. An array name followed by a subscript within square brackets can be used as an `lvalue` and thus as an expression, as described in the grammar; see [Grammar](#) (on page 2447). Unsubscripted array names can be used in only the following contexts:

- 77363 • A parameter in a function definition or function call
- 77364 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see [Grammar](#), on page 2447); if the name used in this context is not an array name, the behavior is undefined

77367 A valid array *index* shall consist of one or more <comma>-separated expressions, similar to the way in which multi-dimensional arrays are indexed in some programming languages. Because *awk* arrays are really one-dimensional, such a <comma>-separated list shall be converted to a single string by concatenating the string values of the separate expressions, each separated from the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be equivalent:

77373 `var[expr1, expr2, ... exprn]`

77374 `var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]`

77375 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is  
 77376 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall  
 77377 not cause that element to exist. Any other reference to a nonexistent array element shall  
 77378 automatically create it.

77379 Comparisons (with the '<', '<=', '!=', '==', '>', and '>=' operators) shall be made  
 77380 numerically if both operands are numeric, if one is numeric and the other has a string value that  
 77381 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise,  
 77382 operands shall be converted to strings as required and a string comparison shall be made using  
 77383 the locale-specific collation sequence. The value of the comparison expression shall be 1 if the  
 77384 relation is true, or 0 if the relation is false.

### 77385 Variables and Special Variables

77386 Variables can be used in an *awk* program by referencing them. With the exception of function  
 77387 parameters (see [User-Defined Functions](#), on page 2446), they are not explicitly declared.  
 77388 Function parameter names shall be local to the function; all other variable names shall be global.  
 77389 The same name shall not be used as both a function parameter name and as the name of a  
 77390 function or a special *awk* variable. The same name shall not be used both as a variable name with  
 77391 global scope and as the name of a function. The same name shall not be used within the same  
 77392 scope both as a scalar variable and as an array. Uninitialized variables, including scalar  
 77393 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized  
 77394 value shall have both a numeric value of zero and a string value of the empty string. Evaluation  
 77395 of variables with an uninitialized value, to either string or numeric, shall be determined by the  
 77396 context in which they are used.

77397 Field variables shall be designated by a '\$' followed by a number or numerical expression. The  
 77398 effect of the field number *expression* evaluating to anything other than a non-negative integer is  
 77399 unspecified; uninitialized variables or string values need not be converted to numeric values in  
 77400 this context. New field variables can be created by assigning a value to them. References to  
 77401 nonexistent fields (that is, fields after \$NF), shall evaluate to the uninitialized value. Such  
 77402 references shall not create new fields. However, assigning to a nonexistent field (for example,  
 77403 \$(NF+2)=5) shall increase the value of NF; create any intervening fields with the uninitialized  
 77404 value; and cause the value of \$0 to be recomputed, with the fields being separated by the value  
 77405 of OFS. Each field variable shall have a string value or an uninitialized value when created.  
 77406 Field variables shall have the uninitialized value when created from \$0 using FS and the variable  
 77407 does not contain any characters. If appropriate, the field variable shall be considered a numeric  
 77408 string (see [Expressions in awk](#), on page 2433).

77409 Implementations shall support the following other special variables that are set by *awk*:

77410 **ARGC** The number of elements in the **ARGV** array.

77411 **ARGV** An array of command line arguments, excluding options and the *program*  
 77412 argument, numbered from zero to **ARGC**-1.

77413 The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As  
 77414 each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the  
 77415 current value of **ARGC**-1, inclusive, as the name of the next input file. Thus,  
 77416 setting an element of **ARGV** to null means that it shall not be treated as an input  
 77417 file. The name '-' indicates the standard input. If an argument matches the format  
 77418 of an *assignment* operand, this argument shall be treated as an *assignment* rather  
 77419 than a *file* argument.

77420	<b>CONVFMT</b>	The <b>printf</b> format for converting numbers to strings (except for output statements, where <b>OFMT</b> is used); "% . 6g" by default.
77421		
77422	<b>ENVIRON</b>	An array representing the value of the environment, as described in the <i>exec</i> functions defined in the System Interfaces volume of POSIX.1-2008. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a <i>numeric string</i> (see <a href="#">Expressions in awk</a> , on page 2433); the array element shall also have its numeric value.
77423		
77424		
77425		
77426		
77427		
77428		
77429		In all cases where the behavior of <i>awk</i> is affected by environment variables (including the environment of any commands that <i>awk</i> executes via the <b>system</b> function or via pipeline redirections with the <b>print</b> statement, the <b>printf</b> statement, or the <b>getline</b> function), the environment used shall be the environment at the time <i>awk</i> began executing; it is implementation-defined whether any modification of <b>ENVIRON</b> affects this environment.
77430		
77431		
77432		
77433		
77434		
77435	<b>FILENAME</b>	A pathname of the current input file. Inside a <b>BEGIN</b> action the value is undefined. Inside an <b>END</b> action the value shall be the name of the last input file processed.
77436		
77437		
77438	<b>FNR</b>	The ordinal number of the current record in the current file. Inside a <b>BEGIN</b> action the value shall be zero. Inside an <b>END</b> action the value shall be the number of the last record processed in the last file processed.
77439		
77440		
77441	<b>FS</b>	Input field separator regular expression; a <space> by default.
77442	<b>NF</b>	The number of fields in the current record. Inside a <b>BEGIN</b> action, the use of <b>NF</b> is undefined unless a <b>getline</b> function without a <i>var</i> argument is executed previously. Inside an <b>END</b> action, <b>NF</b> shall retain the value it had for the last record read, unless a subsequent, redirected, <b>getline</b> function without a <i>var</i> argument is performed prior to entering the <b>END</b> action.
77443		
77444		
77445		
77446		
77447	<b>NR</b>	The ordinal number of the current record from the start of input. Inside a <b>BEGIN</b> action the value shall be zero. Inside an <b>END</b> action the value shall be the number of the last record processed.
77448		
77449		
77450	<b>OFMT</b>	The <b>printf</b> format for converting numbers to strings in output statements (see <a href="#">Output Statements</a> , on page 2442); "% . 6g" by default. The result of the conversion is unspecified if the value of <b>OFMT</b> is not a floating-point format specification.
77451		
77452		
77453	<b>OFS</b>	The <b>print</b> statement output field separator; <space> by default.
77454	<b>ORS</b>	The <b>print</b> statement output record separator; a <newline> by default.
77455	<b>RLENGTH</b>	The length of the string matched by the <b>match</b> function.
77456	<b>RS</b>	The first character of the string value of <b>RS</b> shall be the input record separator; a <newline> by default. If <b>RS</b> contains more than one character, the results are unspecified. If <b>RS</b> is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of <b>FS</b> is.
77457		
77458		
77459		
77460		
77461		
77462	<b>RSTART</b>	The starting position of the string matched by the <b>match</b> function, numbering from 1. This shall always be equivalent to the return value of the <b>match</b> function.
77463		

77464 **SUBSEP** The subscript separator string for multi-dimensional arrays; the default value is  
77465 implementation-defined.

### 77466 Regular Expressions

77467 The *awk* utility shall make use of the extended regular expression notation (see XBD Section 9.4,  
77468 on page 188) except that it shall allow the use of C-language conventions for escaping special  
77469 characters within the EREs, as specified in the table in XBD Chapter 5 (on page 121) ('\\',  
77470 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v') and the following table; these escape sequences  
77471 shall be recognized both inside and outside bracket expressions. Note that records need not be  
77472 separated by <newline> characters and string constants can contain <newline> characters, so  
77473 even the "\\n" sequence is valid in *awk* EREs. Using a <slash> character within an ERE requires  
77474 the escaping shown in the following table.

77475 **Table 4-2** Escape Sequences in *awk*

Escape Sequence	Description	Meaning
\"	<backslash> <quotation-mark>	<quotation-mark> character
\/	<backslash> <slash>	<slash> character
\\ddd	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
\\c	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\\', '\\a', '\\b', '\\f', '\\h', '\\r', '\\t', '\\v').	Undefined

77492 A regular expression can be matched against a specific field or string by using one of the two  
77493 regular expression matching operators, '~' and '!~'. These operators shall interpret their  
77494 right-hand operand as a regular expression and their left-hand operand as a string. If the regular  
77495 expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~'  
77496 expression shall evaluate to a value of 0. (The regular expression matching operation is as  
77497 defined by the term matched in XBD Section 9.1 (on page 181), where a match occurs on any part  
77498 of the string unless the regular expression is limited with the <circumflex> or <dollar-sign>  
77499 special characters.) If the regular expression does not match the string, the '~' expression shall  
77500 evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand  
77501 operand is any expression other than the lexical token ERE, the string value of the expression  
77502 shall be interpreted as an extended regular expression, including the escape conventions  
77503 described above. Note that these same escape conventions shall also be applied in determining  
77504 the value of a string literal (the lexical token STRING), and thus shall be applied a second time  
77505 when a string literal is used in this context.

77506 When an ERE token appears as an expression in any context other than as the right-hand of the  
77507 '~' or '!~' operator or as one of the built-in function arguments described below, the value of  
77508 the resulting expression shall be the equivalent of:

77509 \$0 ~ /ere/

77510 The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function  
 77511 (see [String Functions](#), on page 2444) shall be interpreted as extended regular expressions. These  
 77512 can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner  
 77513 as the right-hand side of the '*~*' or '!~' operator.

77514 An extended regular expression can be used to separate fields by using the **-F ERE** option or by  
 77515 assigning a string containing the expression to the built-in variable **FS**. The default value of the  
 77516 **FS** variable shall be a single <space>. The following describes **FS** behavior:

- 77517 1. If **FS** is a null string, the behavior is unspecified.
- 77518 2. If **FS** is a single character:
  - 77519 a. If **FS** is <space>, skip leading and trailing <blank> characters; fields shall be  
 77520 delimited by sets of one or more <blank> characters.
  - 77521 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single  
 77522 occurrence of *c*.
- 77523 3. Otherwise, the string value of **FS** shall be considered to be an extended regular  
 77524 expression. Each occurrence of a sequence matching the extended regular expression shall  
 77525 delimit fields.

77526 Except for the '*~*' and '!~' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,  
 77527 ERE matching shall be based on input records; that is, record separator characters (the first  
 77528 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the  
 77529 expression, and no expression shall match the record separator character. If the record separator  
 77530 is not <newline>, <newline> characters embedded in the expression can be matched. For the  
 77531 '*~*' and '!~' operators, and in those four built-in functions, ERE matching shall be based on  
 77532 text strings; that is, any character (including <newline> and the record separator) can be  
 77533 embedded in the pattern, and an appropriate pattern shall match any character. However, in all  
 77534 *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text  
 77535 string produces undefined results.

### 77536 Patterns

77537 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or  
 77538 one of the two special patterns **BEGIN** or **END**.

### 77539 Special Patterns

77540 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern  
 77541 shall be matched once and its associated action executed before the first record of input is read—  
 77542 except possibly by use of the **getline** function (see [Input/Output and General Functions](#), on  
 77543 page 2445) in a prior **BEGIN** action—and before command line assignment is done. Each **END**  
 77544 pattern shall be matched once and its associated action executed after the last record of input has  
 77545 been read. These two patterns shall have associated actions.

77546 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns  
 77547 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order  
 77548 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern  
 77549 in a program.

77550 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action  
 77551 contains no **getline** function, *awk* shall exit without reading its input when the last statement in  
 77552 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

77553 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the  
77554 statements in the **END** actions are executed.

### 77555 **Expression Patterns**

77556 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the  
77557 result is true, the pattern shall be considered to match, and the associated action (if any) shall be  
77558 executed. If the result is false, the action shall not be executed.

### 77559 **Pattern Ranges**

77560 A pattern range consists of two expressions separated by a comma; in this case, the action shall  
77561 be performed for all records between a match of the first expression and the following match of  
77562 the second expression, inclusive. At this point, the pattern range can be repeated starting at  
77563 input records subsequent to the end of the matched range.

### 77564 **Actions**

77565 An action is a sequence of statements as shown in the grammar in [Grammar](#) (on page 2447).  
77566 Any single statement can be replaced by a statement list enclosed in curly braces. The  
77567 application shall ensure that statements in a statement list are separated by <newline> or  
77568 <semicolon> characters. Statements in a statement list shall be executed sequentially in the order  
77569 that they appear.

77570 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero  
77571 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement  
77572 following the **else** shall be executed.

77573 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard  
77574 (see [Section 1.1.2](#), on page 2283), except that the Boolean expressions shall be treated as  
77575 described in [Expressions in awk](#) (on page 2433), and except in the case of:

77576 `for (variable in array)`

77577 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of  
77578 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**  
77579 statement occurs outside of a loop, the behavior is undefined.

77580 The **delete** statement shall remove an individual array element. Thus, the following code deletes  
77581 an entire array:

```
77582 for (index in array)
77583     delete array[index]
```

77584 The **next** statement shall cause all further processing of the current input record to be  
77585 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or  
77586 **END** action.

77587 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program  
77588 source and then terminate the program without reading further input. An **exit** statement inside  
77589 an **END** action shall terminate the program without further execution of **END** actions. If an  
77590 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,  
77591 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is  
77592 executed.

77593 **Output Statements**

77594 Both **print** and **printf** statements shall write to standard output by default. The output shall be  
77595 written to the location specified by *output\_redirection* if one is supplied, as follows:

```
77596 > expression
77597 >> expression
77598 | expression
```

77599 In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into  
77600 which to write (for '>' or ">>") or as a command to be executed (for '|'). Using the first two  
77601 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary  
77602 and using the first form, truncating the file. The output then shall be appended to the file. As  
77603 long as the file remains open, subsequent calls in which *expression* evaluates to the same string  
77604 value shall simply append output to the file. The file remains open until the **close** function (see  
77605 [Input/Output and General Functions](#), on page 2445) is called with an expression that evaluates  
77606 to the same string value.

77607 The third form shall write output onto a stream piped to the input of a command. The stream  
77608 shall be created if no stream is currently open with the value of *expression* as its command name.  
77609 The stream created shall be equivalent to one created by a call to the *popen()* function defined in  
77610 the System Interfaces volume of POSIX.1-2008 with the value of *expression* as the *command*  
77611 argument and a value of *w* as the *mode* argument. As long as the stream remains open,  
77612 subsequent calls in which *expression* evaluates to the same string value shall write output to the  
77613 existing stream. The stream shall remain open until the **close** function (see [Input/Output and](#)  
77614 [General Functions](#), on page 2445) is called with an expression that evaluates to the same string  
77615 value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in  
77616 the System Interfaces volume of POSIX.1-2008.

77617 As described in detail by the grammar in [Grammar](#) (on page 2447), these output statements shall  
77618 take a <comma>-separated list of *expressions* referred to in the grammar by the non-terminal  
77619 symbols **expr\_list**, **print\_expr\_list**, or **print\_expr\_list\_opt**. This list is referred to here as the  
77620 *expression list*, and each member is referred to as an *expression argument*.

77621 The **print** statement shall write the value of each expression argument onto the indicated output  
77622 stream separated by the current output field separator (see variable **OFS** above), and terminated  
77623 by the output record separator (see variable **ORS** above). All expression arguments shall be  
77624 taken as strings, being converted if necessary; this conversion shall be as described in  
77625 [Expressions in awk](#) (on page 2433), with the exception that the **printf** format in **OFMT** shall be  
77626 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole  
77627 input record (\$0).

77628 The **printf** statement shall produce output based on a notation similar to the File Format  
77629 Notation used to describe file formats in this volume of POSIX.1-2008 (see [XBD Chapter 5](#), on  
77630 page 121). Output shall be produced as specified with the first *expression* argument as the string  
77631 *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the  
77632 following exceptions:

- 77633 1. The *format* shall be an actual character string rather than a graphical representation.  
77634 Therefore, it cannot contain empty character positions. The <space> in the *format* string,  
77635 in any context other than a *flag* of a conversion specification, shall be treated as an  
77636 ordinary character that is copied to the output.
- 77637 2. If the character set contains a 'Δ' character and that character appears in the *format*  
77638 string, it shall be treated as an ordinary character that is copied to the output.

- 77639 3. The *escape sequences* beginning with a <backslash> character shall be treated as sequences  
77640 of ordinary characters that are copied to the output. Note that these same sequences shall  
77641 be interpreted lexically by *awk* when they appear in literal strings, but they shall not be  
77642 treated specially by the **printf** statement.
- 77643 4. A *field width* or *precision* can be specified as the ' \* ' character instead of a digit string. In  
77644 this case the next argument from the expression list shall be fetched and its numeric value  
77645 taken as the field width or precision.
- 77646 5. The implementation shall not precede or follow output from the *d* or *u* conversion  
77647 specifier characters with <blank> characters not specified by the *format* string.
- 77648 6. The implementation shall not precede output from the *o* conversion specifier character  
77649 with leading zeros not specified by the *format* string.
- 77650 7. For the *c* conversion specifier character: if the argument has a numeric value, the  
77651 character whose encoding is that value shall be output. If the value is zero or is not the  
77652 encoding of any character in the character set, the behavior is undefined. If the argument  
77653 does not have a numeric value, the first character of the string value shall be output; if the  
77654 string does not contain any characters, the behavior is undefined.
- 77655 8. For each conversion specification that consumes an argument, the next expression  
77656 argument shall be evaluated. With the exception of the *c* conversion specifier character,  
77657 the value shall be converted (according to the rules specified in [Expressions in awk](#), on  
77658 page 2433) to the appropriate type for the conversion specification.
- 77659 9. If there are insufficient expression arguments to satisfy all the conversion specifications in  
77660 the *format* string, the behavior is undefined.
- 77661 10. If any character sequence in the *format* string begins with a ' % ' character, but does not  
77662 form a valid conversion specification, the behavior is unspecified.

77663 Both **print** and **printf** can output at least {LINE\_MAX} bytes.

## 77664 Functions

77665 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and  
77666 general.

### 77667 Arithmetic Functions

77668 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see [Section 1.1.2](#),  
77669 on page 2283). The behavior is undefined in cases where the ISO C standard specifies that an  
77670 error be returned or that the behavior is undefined. Although the grammar (see [Grammar](#), on  
77671 page 2447) permits built-in functions to appear with no arguments or parentheses, unless the  
77672 argument or parentheses are indicated as optional in the following list (by displaying them  
77673 within the " [ ] " brackets), such use is undefined.

- 77674 **atan2**(*y,x*) Return arctangent of *y/x* in radians in the range  $[-\pi, \pi]$ .
- 77675 **cos**(*x*) Return cosine of *x*, where *x* is in radians.
- 77676 **sin**(*x*) Return sine of *x*, where *x* is in radians.
- 77677 **exp**(*x*) Return the exponential function of *x*.
- 77678 **log**(*x*) Return the natural logarithm of *x*.

77679	<b>sqrt</b> ( <i>x</i> )	Return the square root of <i>x</i> .
77680	<b>int</b> ( <i>x</i> )	Return the argument truncated to an integer. Truncation shall be toward 0 when <i>x</i> >0.
77681		
77682	<b>rand</b> ()	Return a random number <i>n</i> , such that $0 \leq n < 1$ .
77683	<b>srand</b> ([ <i>expr</i> ])	Set the seed value for <i>rand</i> to <i>expr</i> or use the time of day if <i>expr</i> is omitted. The previous seed value shall be returned.
77684		

77685 **String Functions**

77686 The string functions in the following list shall be supported. Although the grammar (see  
77687 [Grammar](#), on page 2447) permits built-in functions to appear with no arguments or parentheses,  
77688 unless the argument or parentheses are indicated as optional in the following list (by displaying  
77689 them within the " [ ] " brackets), such use is undefined.

77690	<b>gsub</b> ( <i>ere</i> , <i>repl</i> [, <i>in</i> ])	
77691		Behave like <b>sub</b> (see below), except that it shall replace all occurrences of the
77692		regular expression (like the <i>ed</i> utility global substitute) in \$0 or in the <i>in</i> argument,
77693		when specified.
77694	<b>index</b> ( <i>s</i> , <i>t</i> )	Return the position, in characters, numbering from 1, in string <i>s</i> where string <i>t</i> first
77695		occurs, or zero if it does not occur at all.
77696	<b>length</b> ([ <i>s</i> ])	Return the length, in characters, of its argument taken as a string, or of the whole
77697		record, \$0, if there is no argument.
77698	<b>match</b> ( <i>s</i> , <i>ere</i> )	Return the position, in characters, numbering from 1, in string <i>s</i> where the
77699		extended regular expression <i>ere</i> occurs, or zero if it does not occur at all. RSTART
77700		shall be set to the starting position (which is the same as the returned value), zero
77701		if no match is found; RLENGTH shall be set to the length of the matched string, -1
77702		if no match is found.
77703	<b>split</b> ( <i>s</i> , <i>a</i> [, <i>fs</i> ])	
77704		Split the string <i>s</i> into array elements <i>a</i> [1], <i>a</i> [2], ..., <i>a</i> [ <i>n</i> ], and return <i>n</i> . All elements
77705		of the array shall be deleted before the split is performed. The separation shall be
77706		done with the ERE <i>fs</i> or with the field separator <b>FS</b> if <i>fs</i> is not given. Each array
77707		element shall have a string value when created and, if appropriate, the array
77708		element shall be considered a numeric string (see <a href="#">Expressions in awk</a> , on page
77709		2433). The effect of a null string as the value of <i>fs</i> is unspecified.
77710	<b>sprintf</b> ( <i>fmt</i> , <i>expr</i> , <i>expr</i> , ...)	
77711		Format the expressions according to the <b>printf</b> format given by <i>fmt</i> and return the
77712		resulting string.
77713	<b>sub</b> ( <i>ere</i> , <i>repl</i> [, <i>in</i> ])	
77714		Substitute the string <i>repl</i> in place of the first instance of the extended regular
77715		expression <i>ERE</i> in string <i>in</i> and return the number of substitutions. An
77716		<ampersand> (' & ') appearing in the string <i>repl</i> shall be replaced by the string
77717		from <i>in</i> that matches the ERE. An <ampersand> preceded with a <backslash> shall
77718		be interpreted as the literal <ampersand> character. An occurrence of two
77719		consecutive <backslash> characters shall be interpreted as just a single literal
77720		<backslash> character. Any other occurrence of a <backslash> (for example,
77721		preceding any other character) shall be treated as a literal <backslash> character.
77722		Note that if <i>repl</i> is a string literal (the lexical token <b>STRING</b> ; see <a href="#">Grammar</a> , on page
77723		2447), the handling of the <ampersand> character occurs after any lexical

- 77724 processing, including any lexical <backslash>-escape sequence processing. If *in* is  
 77725 specified and it is not an lvalue (see [Expressions in awk](#), on page 2433), the  
 77726 behavior is undefined. If *in* is omitted, *awk* shall use the current record (\$0) in its  
 77727 place.
- 77728 **substr**(*s*, *m*[, *n* ])  
 77729 Return the at most *n*-character substring of *s* that begins at position *m*, numbering  
 77730 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,  
 77731 the length of the substring shall be limited by the length of the string *s*.
- 77732 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter  
 77733 specified to have a **tolower** mapping by the *LC\_CTYPE* category of the current  
 77734 locale shall be replaced in the returned string by the lowercase letter specified by  
 77735 the mapping. Other characters in *s* shall be unchanged in the returned string.
- 77736 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter  
 77737 specified to have a **toupper** mapping by the *LC\_CTYPE* category of the current  
 77738 locale is replaced in the returned string by the uppercase letter specified by the  
 77739 mapping. Other characters in *s* are unchanged in the returned string.
- 77740 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued  
 77741 expression that is a regular expression as defined in [Regular Expressions](#) (on page 2439).
- 77742 **Input/Output and General Functions**
- 77743 The input/output and general functions are:
- 77744 **close**(*expression*)  
 77745 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with  
 77746 the same string-valued *expression*. The limit on the number of open *expression*  
 77747 arguments is implementation-defined. If the close was successful, the function  
 77748 shall return zero; otherwise, it shall return non-zero.
- 77749 *expression* | **getline** [*var*]  
 77750 Read a record of input from a stream piped from the output of a command. The  
 77751 stream shall be created if no stream is currently open with the value of *expression* as  
 77752 its command name. The stream created shall be equivalent to one created by a call  
 77753 to the *popen*() function with the value of *expression* as the *command* argument and a  
 77754 value of *r* as the *mode* argument. As long as the stream remains open, subsequent  
 77755 calls in which *expression* evaluates to the same string value shall read subsequent  
 77756 records from the stream. The stream shall remain open until the **close** function is  
 77757 called with an expression that evaluates to the same string value. At that time, the  
 77758 stream shall be closed as if by a call to the *pclose*() function. If *var* is omitted, \$0 and  
 77759 NF shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 77760 a numeric string (see [Expressions in awk](#), on page 2433).
- 77761 The **getline** operator can form ambiguous constructs when there are  
 77762 unparenthesized operators (including concatenate) to the left of the ' | ' (to the  
 77763 beginning of the expression containing **getline**). In the context of the ' \$ ' operator,  
 77764 ' | ' shall behave as if it had a lower precedence than ' \$ '. The result of evaluating  
 77765 other operators is unspecified, and conforming applications shall parenthesize  
 77766 properly all such usages.
- 77767 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall  
 77768 set the NF, NR, and FNR variables.

77769 **getline** *var* Set variable *var* to the next input record from the current input file and, if  
 77770 appropriate, *var* shall be considered a numeric string (see [Expressions in awk](#), on  
 77771 page 2433). This form of **getline** shall set the **FNR** and **NR** variables.

77772 **getline** [*var*] < *expression*  
 77773 Read the next record of input from a named file. The *expression* shall be evaluated  
 77774 to produce a string that is used as a pathname. If the file of that name is not  
 77775 currently open, it shall be opened. As long as the stream remains open, subsequent  
 77776 calls in which *expression* evaluates to the same string value shall read subsequent  
 77777 records from the file. The file shall remain open until the **close** function is called  
 77778 with an expression that evaluates to the same string value. If *var* is omitted, **\$0** and  
 77779 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 77780 a numeric string (see [Expressions in awk](#), on page 2433).

77781 The **getline** operator can form ambiguous constructs when there are  
 77782 unparenthesized binary operators (including concatenate) to the right of the '<'  
 77783 (up to the end of the expression containing the **getline**). The result of evaluating  
 77784 such a construct is unspecified, and conforming applications shall parenthesize  
 77785 properly all such usages.

77786 **system**(*expression*)  
 77787 Execute the command given by *expression* in a manner equivalent to the *system*()  
 77788 function defined in the System Interfaces volume of POSIX.1-2008 and return the  
 77789 exit status of the command.

77790 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

77791 Where strings are used as the name of a file or pipeline, the application shall ensure that the  
 77792 strings are textually identical. The terminology "same string value" implies that "equivalent  
 77793 strings", even those that differ only by <space> characters, represent different files.

#### 77794 User-Defined Functions

77795 The *awk* language also provides user-defined functions. Such functions can be defined as:

```
77796 function name([parameter, ...]) { statements }
```

77797 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its  
 77798 definition. The scope of a function is global.

77799 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an  
 77800 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is  
 77801 passed as a parameter that the function uses as an array. Function parameters shall be passed by  
 77802 value if scalar and by reference if array name.

77803 The number of parameters in the function definition need not match the number of parameters  
 77804 in the function call. Excess formal parameters can be used as local variables. If fewer arguments  
 77805 are supplied in a function call than are in the function definition, the extra parameters that are  
 77806 used in the function body as scalars shall evaluate to the uninitialized value until they are  
 77807 otherwise initialized, and the extra parameters that are used in the function body as arrays shall  
 77808 be treated as uninitialized arrays where each element evaluates to the uninitialized value until  
 77809 otherwise initialized.

77810 When invoking a function, no white space can be placed between the function name and the  
 77811 opening parenthesis. Function calls can be nested and recursive calls can be made upon  
 77812 functions. Upon return from any nested or recursive function call, the values of all of the calling  
 77813 function's parameters shall be unchanged, except for array parameters passed by reference. The

77814 **return** statement can be used to return a value. If a **return** statement appears outside of a  
77815 function definition, the behavior is undefined.

77816 In the function definition, <newline> characters shall be optional before the opening brace and  
77817 after the closing brace. Function definitions can appear anywhere in the program where a  
77818 *pattern-action* pair is allowed.

## 77819 Grammar

77820 The grammar in this section and the lexical conventions in the following section shall together  
77821 describe the syntax for *awk* programs. The general conventions for this style of grammar are  
77822 described in Section 1.3 (on page 2287). A valid program can be represented as the non-terminal  
77823 symbol *program* in the grammar. This formal syntax shall take precedence over the preceding  
77824 text syntax description.

```

77825 %token NAME NUMBER STRING ERE
77826 %token FUNC_NAME /* Name followed by '(' without white space. */

77827 /* Keywords */
77828 %token Begin End
77829 /* 'BEGIN' 'END' */

77830 %token Break Continue Delete Do Else
77831 /* 'break' 'continue' 'delete' 'do' 'else' */

77832 %token Exit For Function If In
77833 /* 'exit' 'for' 'function' 'if' 'in' */

77834 %token Next Print Printf Return While
77835 /* 'next' 'print' 'printf' 'return' 'while' */

77836 /* Reserved function names */
77837 %token BUILTIN_FUNC_NAME
77838 /* One token for the following:
77839 * atan2 cos sin exp log sqrt int rand srand
77840 * gsub index length match split sprintf sub
77841 * substr tolower toupper close system
77842 */

77843 %token GETLINE
77844 /* Syntactically different from other built-ins. */

77845 /* Two-character tokens. */
77846 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
77847 /* '+=' '-=' '*=' '/=' '%=' '^=' */

77848 %token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
77849 /* '||' '&&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */

77850 /* One-character tokens. */
77851 %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
77852 %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

77853 %start program
77854 %%

77855 program : item_list
77856 | actionless_item_list
77857 ;

```

```

77858     item_list           : newline_opt
77859                           | actionless_item_list item terminator
77860                           | item_list           item terminator
77861                           | item_list           action terminator
77862                           ;

77863     actionless_item_list : item_list           pattern terminator
77864                           | actionless_item_list pattern terminator
77865                           ;

77866     item                 : pattern action
77867                           | Function NAME      '(' param_list_opt ')'
77868                             newline_opt action
77869                           | Function FUNC_NAME '(' param_list_opt ')'
77870                             newline_opt action
77871                           ;

77872     param_list_opt       : /* empty */
77873                           | param_list
77874                           ;

77875     param_list           : NAME
77876                           | param_list ',' NAME
77877                           ;

77878     pattern              : Begin
77879                           | End
77880                           | expr
77881                           | expr ',' newline_opt expr
77882                           ;

77883     action               : '{' newline_opt      '}'
77884                           | '{' newline_opt terminated_statement_list '}'
77885                           | '{' newline_opt unterminated_statement_list '}'
77886                           ;

77887     terminator           : terminator ';'
77888                           | terminator NEWLINE
77889                           | ';'
77890                           | NEWLINE
77891                           ;

77892     terminated_statement_list : terminated_statement
77893                           | terminated_statement_list terminated_statement
77894                           ;

77895     unterminated_statement_list : unterminated_statement
77896                           | terminated_statement_list unterminated_statement
77897                           ;

77898     terminated_statement : action newline_opt
77899                           | If '(' expr ')' newline_opt terminated_statement
77900                           | If '(' expr ')' newline_opt terminated_statement
77901                             Else newline_opt terminated_statement
77902                           | While '(' expr ')' newline_opt terminated_statement
77903                           | For '(' simple_statement_opt ';'
77904                             expr_opt ';' simple_statement_opt ')' newline_opt

```

```

77905         terminated_statement
77906     | For '(' NAME In NAME ')' newline_opt
77907         terminated_statement
77908     | ';' newline_opt
77909     | terminatable_statement NEWLINE newline_opt
77910     | terminatable_statement ';' newline_opt
77911     ;

77912 unterminated_statement : terminatable_statement
77913     | If '(' expr ')' newline_opt unterminated_statement
77914     | If '(' expr ')' newline_opt terminated_statement
77915     | Else newline_opt unterminated_statement
77916     | While '(' expr ')' newline_opt unterminated_statement
77917     | For '(' simple_statement_opt ';'
77918     |   expr_opt ';' simple_statement_opt ')' newline_opt
77919     |   unterminated_statement
77920     | For '(' NAME In NAME ')' newline_opt
77921     |   unterminated_statement
77922     ;

77923 terminatable_statement : simple_statement
77924     | Break
77925     | Continue
77926     | Next
77927     | Exit expr_opt
77928     | Return expr_opt
77929     | Do newline_opt terminated_statement While '(' expr ')'
77930     ;

77931 simple_statement_opt : /* empty */
77932     | simple_statement
77933     ;

77934 simple_statement : Delete NAME '[' expr_list '['
77935     | expr
77936     | print_statement
77937     ;

77938 print_statement : simple_print_statement
77939     | simple_print_statement output_redirection
77940     ;

77941 simple_print_statement : Print print_expr_list_opt
77942     | Print '(' multiple_expr_list ')'
77943     | Printf print_expr_list
77944     | Printf '(' multiple_expr_list ')'
77945     ;

77946 output_redirection : '>' expr
77947     | APPEND expr
77948     | '|' expr
77949     ;

77950 expr_list_opt : /* empty */
77951     | expr_list
77952     ;

```

```

77953     expr_list      : expr
77954                       | multiple_expr_list
77955                       ;

77956     multiple_expr_list : expr ',' newline_opt expr
77957                       | multiple_expr_list ',' newline_opt expr
77958                       ;

77959     expr_opt        : /* empty */
77960                       | expr
77961                       ;

77962     expr            : unary_expr
77963                       | non_unary_expr
77964                       ;

77965     unary_expr      : '+' expr
77966                       | '-' expr
77967                       | unary_expr '^'      expr
77968                       | unary_expr '*'      expr
77969                       | unary_expr '/'      expr
77970                       | unary_expr '%'      expr
77971                       | unary_expr '+'      expr
77972                       | unary_expr '-'      expr
77973                       | unary_expr          non_unary_expr
77974                       | unary_expr '<'      expr
77975                       | unary_expr LE      expr
77976                       | unary_expr NE      expr
77977                       | unary_expr EQ      expr
77978                       | unary_expr '>'      expr
77979                       | unary_expr GE      expr
77980                       | unary_expr '~'      expr
77981                       | unary_expr NO_MATCH expr
77982                       | unary_expr In NAME
77983                       | unary_expr AND newline_opt expr
77984                       | unary_expr OR  newline_opt expr
77985                       | unary_expr '?' expr ':' expr
77986                       | unary_input_function
77987                       ;

77988     non_unary_expr  : '(' expr ')'
77989                       | '!' expr
77990                       | non_unary_expr '^'      expr
77991                       | non_unary_expr '*'      expr
77992                       | non_unary_expr '/'      expr
77993                       | non_unary_expr '%'      expr
77994                       | non_unary_expr '+'      expr
77995                       | non_unary_expr '-'      expr
77996                       | non_unary_expr          non_unary_expr
77997                       | non_unary_expr '<'      expr
77998                       | non_unary_expr LE      expr
77999                       | non_unary_expr NE      expr
78000                       | non_unary_expr EQ      expr
78001                       | non_unary_expr '>'      expr

```

```

78002         | non_unary_expr GE          expr
78003         | non_unary_expr '~'          expr
78004         | non_unary_expr NO_MATCH expr
78005         | non_unary_expr In NAME
78006         | '(' multiple_expr_list ')' In NAME
78007         | non_unary_expr AND newline_opt expr
78008         | non_unary_expr OR  newline_opt expr
78009         | non_unary_expr '?' expr ':' expr
78010         | NUMBER
78011         | STRING
78012         | lvalue
78013         | ERE
78014         | lvalue INCR
78015         | lvalue DECR
78016         | INCR lvalue
78017         | DECR lvalue
78018         | lvalue POW_ASSIGN expr
78019         | lvalue MOD_ASSIGN expr
78020         | lvalue MUL_ASSIGN expr
78021         | lvalue DIV_ASSIGN expr
78022         | lvalue ADD_ASSIGN expr
78023         | lvalue SUB_ASSIGN expr
78024         | lvalue '=' expr
78025         | FUNC_NAME '(' expr_list_opt ')'
78026         | /* no white space allowed before '(' */
78027         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
78028         | BUILTIN_FUNC_NAME
78029         | non_unary_input_function
78030         ;

78031     print_expr_list_opt : /* empty */
78032         | print_expr_list
78033         ;

78034     print_expr_list : print_expr
78035         | print_expr_list ',' newline_opt print_expr
78036         ;

78037     print_expr : unary_print_expr
78038         | non_unary_print_expr
78039         ;

78040     unary_print_expr : '+' print_expr
78041         | '-' print_expr
78042         | unary_print_expr '^' print_expr
78043         | unary_print_expr '*' print_expr
78044         | unary_print_expr '/' print_expr
78045         | unary_print_expr '%' print_expr
78046         | unary_print_expr '+' print_expr
78047         | unary_print_expr '-' print_expr
78048         | unary_print_expr non_unary_print_expr
78049         | unary_print_expr '~' print_expr
78050         | unary_print_expr NO_MATCH print_expr
78051         | unary_print_expr In NAME

```

```

78052         | unary_print_expr AND newline_opt print_expr
78053         | unary_print_expr OR  newline_opt print_expr
78054         | unary_print_expr '?' print_expr ':' print_expr
78055         ;

78056 non_unary_print_expr : '(' expr ')'
78057         | '!' print_expr
78058         | non_unary_print_expr '^'      print_expr
78059         | non_unary_print_expr '*'      print_expr
78060         | non_unary_print_expr '/'      print_expr
78061         | non_unary_print_expr '%'      print_expr
78062         | non_unary_print_expr '+'      print_expr
78063         | non_unary_print_expr '-'      print_expr
78064         | non_unary_print_expr         non_unary_print_expr
78065         | non_unary_print_expr '~'      print_expr
78066         | non_unary_print_expr NO_MATCH print_expr
78067         | non_unary_print_expr In NAME
78068         | '(' multiple_expr_list ')' In NAME
78069         | non_unary_print_expr AND newline_opt print_expr
78070         | non_unary_print_expr OR  newline_opt print_expr
78071         | non_unary_print_expr '?' print_expr ':' print_expr
78072         | NUMBER
78073         | STRING
78074         | lvalue
78075         | ERE
78076         | lvalue INCR
78077         | lvalue DECR
78078         | INCR lvalue
78079         | DECR lvalue
78080         | lvalue POW_ASSIGN print_expr
78081         | lvalue MOD_ASSIGN print_expr
78082         | lvalue MUL_ASSIGN print_expr
78083         | lvalue DIV_ASSIGN print_expr
78084         | lvalue ADD_ASSIGN print_expr
78085         | lvalue SUB_ASSIGN print_expr
78086         | lvalue '=' print_expr
78087         | FUNC_NAME '(' expr_list_opt ')'
78088         /* no white space allowed before '(' */
78089         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
78090         | BUILTIN_FUNC_NAME
78091         ;

78092 lvalue      : NAME
78093         | NAME '[' expr_list ']'
78094         | '$' expr
78095         ;

78096 non_unary_input_function : simple_get
78097         | simple_get '<' expr
78098         | non_unary_expr '|' simple_get
78099         ;

78100 unary_input_function : unary_expr '|' simple_get
78101         ;

```

```

78102     simple_get      : GETLINE
78103                       | GETLINE lvalue
78104                       ;
78105     newline_opt     : /* empty */
78106                       | newline_opt NEWLINE
78107                       ;

```

78108 This grammar has several ambiguities that shall be resolved as follows:

- 78109 • Operator precedence and associativity shall be as described in Table 4-1 (on page 2433).
- 78110 • In case of ambiguity, an **else** shall be associated with the most immediately preceding **if**
- 78111 that would satisfy the grammar.
- 78112 • In some contexts, a <slash> ('/') that is used to surround an ERE could also be the
- 78113 division operator. This shall be resolved in such a way that wherever the division operator
- 78114 could appear, a <slash> is assumed to be the division operator. (There is no unary division
- 78115 operator.)

78116 Each expression in an *awk* program shall conform to the precedence and associativity rules, even

78117 when this is not needed to resolve an ambiguity. For example, because '\$' has higher

78118 precedence than '++', the string "\$x++--" is not a valid *awk* expression, even though it is

78119 unambiguously parsed by the grammar as "\$ (x++) --".

78120 One convention that might not be obvious from the formal grammar is where <newline>

78121 characters are acceptable. There are several obvious placements such as terminating a statement,

78122 and a <backslash> can be used to escape <newline> characters between any lexical tokens. In

78123 addition, <newline> characters without <backslash> characters can follow a comma, an open

78124 brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else**

78125 keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

78126 { print $1,
78127         $2 }

```

## 78128 Lexical Conventions

78129 The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as

78130 follows:

- 78131 1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a
- 78132 given point.
- 78133 2. A comment shall consist of any characters beginning with the <number-sign> character
- 78134 and terminated by, but excluding the next occurrence of, a <newline>. Comments shall
- 78135 have no effect, except to delimit lexical tokens.
- 78136 3. The <newline> shall be recognized as the token **NEWLINE**.
- 78137 4. A <backslash> character immediately followed by a <newline> shall have no effect.
- 78138 5. The token **STRING** shall represent a string constant. A string constant shall begin with
- 78139 the character ' "'. Within a string constant, a <backslash> character shall be considered
- 78140 to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 121)
- 78141 ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences
- 78142 in Table 4-2 (on page 2439) shall be recognized. A <newline> shall not occur within a
- 78143 string constant. A string constant shall be terminated by the first unescaped occurrence of
- 78144 the character ' "' after the one that begins the string constant. The value of the string

- 78145 shall be the sequence of all unescaped characters and values of escape sequences  
 78146 between, but not including, the two delimiting ' ' characters.
- 78147 6. The token **ERE** represents an extended regular expression constant. An ERE constant  
 78148 shall begin with the <slash> character. Within an ERE constant, a <backslash> character  
 78149 shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5  
 78150 (on page 121). In addition, the escape sequences in Table 4-2 (on page 2439) shall be  
 78151 recognized. The application shall ensure that a <newline> does not occur within an ERE  
 78152 constant. An ERE constant shall be terminated by the first unescaped occurrence of the  
 78153 <slash> character after the one that begins the ERE constant. The extended regular  
 78154 expression represented by the ERE constant shall be the sequence of all unescaped  
 78155 characters and values of escape sequences between, but not including, the two delimiting  
 78156 <slash> characters.
- 78157 7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE**  
 78158 tokens.
- 78159 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall  
 78160 either be equivalent to the **decimal-floating-constant** token as specified by the ISO C  
 78161 standard, or it shall be a sequence of decimal digits and shall be evaluated as an integer  
 78162 constant in decimal. In addition, implementations may accept numeric constants with the  
 78163 form and numeric value equivalent to the **hexadecimal-constant** and **hexadecimal-**  
 78164 **floating-constant** tokens as specified by the ISO C standard.
- 78165 If the value is too large or too small to be representable (see Section 1.1.2, on page 2283),  
 78166 the behavior is undefined.
- 78167 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see  
 78168 XBD Section 6.1, on page 125), beginning with an <underscore> or alphabetic character,  
 78169 shall be considered a word.
- 78170 10. The following words are keywords that shall be recognized as individual tokens; the  
 78171 name of the token is the same as the keyword:
- |       |                 |               |             |                 |              |               |
|-------|-----------------|---------------|-------------|-----------------|--------------|---------------|
| 78172 | <b>BEGIN</b>    | <b>delete</b> | <b>END</b>  | <b>function</b> | <b>in</b>    | <b>printf</b> |
| 78173 | <b>break</b>    | <b>do</b>     | <b>exit</b> | <b>getline</b>  | <b>next</b>  | <b>return</b> |
| 78174 | <b>continue</b> | <b>else</b>   | <b>for</b>  | <b>if</b>       | <b>print</b> | <b>while</b>  |
- 78175 11. The following words are names of built-in functions and shall be recognized as the token  
 78176 **BUILTIN\_FUNC\_NAME**:
- |       |              |               |              |                |                |                |
|-------|--------------|---------------|--------------|----------------|----------------|----------------|
| 78177 | <b>atan2</b> | <b>gsub</b>   | <b>log</b>   | <b>split</b>   | <b>sub</b>     | <b>toupper</b> |
| 78178 | <b>close</b> | <b>index</b>  | <b>match</b> | <b>sprintf</b> | <b>substr</b>  |                |
| 78179 | <b>cos</b>   | <b>int</b>    | <b>rand</b>  | <b>sqrt</b>    | <b>system</b>  |                |
| 78180 | <b>exp</b>   | <b>length</b> | <b>sin</b>   | <b>srand</b>   | <b>tolower</b> |                |
- 78181 The above-listed keywords and names of built-in functions are considered reserved  
 78182 words.
- 78183 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in  
 78184 function and is not followed immediately (without any delimiters) by the ' ( ' character.
- 78185 13. The token **FUNC\_NAME** shall consist of a word that is not a keyword or a name of a  
 78186 built-in function, followed immediately (without any delimiters) by the ' ( ' character.  
 78187 The ' ( ' character shall not be included as part of the token.

78188 14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	NO_MATCH	!~
SUB_ASSIGN	--	EQ	==
MUL_ASSIGN	*=	LE	<=
DIV_ASSIGN	/=	GE	>=
MOD_ASSIGN	%=	NE	!=
POW_ASSIGN	^=	INCR	++
OR		DECR	--
AND	&&	APPEND	>>

78198 15. The following single characters shall be recognized as tokens whose names are the  
78199 character:

78200 <newline> { } ( ) [ ] , ; + - \* % ^ ! > < | ? : ~ \$ =

78201 There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV\_ASSIGN**.  
78202 When an input sequence begins with a <slash> character in any syntactic context where the  
78203 token **'/'** or **DIV\_ASSIGN** could appear as the next token in a valid program, the longer of  
78204 those two tokens that can be recognized shall be recognized. In any other syntactic context  
78205 where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be  
78206 recognized.

#### 78207 EXIT STATUS

78208 The following exit values shall be returned:

78209 0 All input files were processed successfully.

78210 >0 An error occurred.

78211 The exit status can be altered within the program by using an **exit** expression.

#### 78212 CONSEQUENCES OF ERRORS

78213 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a  
78214 diagnostic message to standard error and terminate without any further action.

78215 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*  
78216 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

#### 78217 APPLICATION USAGE

78218 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in  
78219 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with  
78220 bytes.

78221 Because the concatenation operation is represented by adjacent expressions rather than an  
78222 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation  
78223 precedence.

#### 78224 EXAMPLES

78225 The *awk* program specified in the command line is most easily specified within single-quotes (for  
78226 example, *'program'*) for applications using *sh*, because *awk* programs commonly contain  
78227 characters that are special to the shell, including double-quotes. In the cases where an *awk*  
78228 program contains single-quote characters, it is usually easiest to specify most of the program as  
78229 strings within single-quotes concatenated by the shell with quoted single-quote characters. For  
78230 example:

78231 `awk '/'\''/ { print "quote:", $0 }'`

78232 prints all lines from the standard input containing a single-quote character, prefixed with *quote*..

78233 The following are examples of simple *awk* programs:

- 78234 1. Write to the standard output all input lines for which field 3 is greater than 5:

78235 `$3 > 5`

- 78236 2. Write every tenth line:

78237 `(NR % 10) == 0`

- 78238 3. Write any line with a substring matching the regular expression:

78239 `/(G|D)(2[0-9][[:alpha:]]*)/`

- 78240 4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits  
78241 and characters. This example uses character classes **digit** and **alpha** to match language-  
78242 independent digit and alphabetic characters respectively:

78243 `/(G|D)([[:digit:]][[:alpha:]]*)/`

- 78244 5. Write any line in which the second field matches the regular expression and the fourth  
78245 field does not:

78246 `$2 ~ /xyz/ && $4 !~ /xyz/`

- 78247 6. Write any line in which the second field contains a <backslash>:

78248 `$2 ~ /\`

- 78249 7. Write any line in which the second field contains a <backslash>. Note that  
78250 <backslash>-escapes are interpreted twice; once in lexical processing of the string and  
78251 once in processing the regular expression:

78252 `$2 ~ "\\`

- 78253 8. Write the second to the last and the last field in each line. Separate the fields by a <colon>:

78254 `{OFS=":";print $(NF-1), $NF}`

- 78255 9. Write the line number and number of fields in each line. The three strings representing  
78256 the line number, the <colon>, and the number of fields are concatenated and that string is  
78257 written to standard output:

78258 `{print NR ":" NF}`

- 78259 10. Write lines longer than 72 characters:

78260 `length($0) > 72`

- 78261 11. Write the first two fields in opposite order separated by **OFS**:

78262 `{ print $2, $1 }`

- 78263 12. Same, with input fields separated by a <comma> or <space> and <tab> characters, or  
78264 both:

78265 `BEGIN { FS = ", [ \t]*|[ \t]+" }`

78266 `{ print $2, $1 }`

- 78267 13. Add up the first column, print sum, and average:

78268 `{s += $1 }`

78269 `END {print "sum is ", s, " average is", s/NR}`

- 78270 14. Write fields in reverse order, one per line (many lines out for each line in):
- ```
78271 { for (i = NF; i > 0; --i) print $i }
```
- 78272 15. Write all lines between occurrences of the strings **start** and **stop**:
- ```
78273 /start/, /stop/
```
- 78274 16. Write all lines whose first field is different from the previous one:
- ```
78275 $1 != prev { print; prev = $1 }
```
- 78276 17. Simulate *echo*:
- ```
78277 BEGIN {
78278     for (i = 1; i < ARGV; ++i)
78279         printf("%s%s", ARGV[i], i==ARGV-1?"\n":" ")
78280 }
```
- 78281 18. Write the path prefixes contained in the *PATH* environment variable, one per line:
- ```
78282 BEGIN {
78283     n = split (ENVIRON["PATH"], path, ":")
78284     for (i = 1; i <= n; ++i)
78285         print path[i]
78286 }
```
- 78287 19. If there is a file named **input** containing page headers of the form:
- ```
78288 Page #
```
- 78289 and a file named **program** that contains:
- ```
78290 /Page/    { $2 = n++; }
78291           { print }
```
- 78292 then the command line:
- ```
78293 awk -f program n=5 input
```
- 78294 prints the file **input**, filling in page numbers starting at 5.

**RATIONALE**

78295 This description is based on the new *awk*, "nawk", (see the referenced *The AWK Programming*  
 78296 *Language*), which introduced a number of new features to the historical *awk*:

- 78298 1. New keywords: **delete**, **do**, **function**, **return**
- 78299 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
- 78300 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 78301 4. New expression operators: **?**, **:**, **..**, **^**
- 78302 5. The **FS** variable and the third argument to **split**, now treated as extended regular  
 78303 expressions.
- 78304 6. The operator precedence, changed to more closely match the C language. Two examples  
 78305 of code that operate differently are:

```
78306 while ( n /= 10 > 1) ...
78307 if (!"wk" ~ /bwk/) ...
```

78308 Several features have been added based on newer implementations of *awk*:

- 78309 • Multiple instances of `-f progfile` are permitted.
- 78310 • The new option `-v assignment`.
- 78311 • The new predefined variable `ENVIRON`.
- 78312 • New built-in functions `toupper` and `tolower`.
- 78313 • More formatting capabilities are added to `printf` to match the ISO C standard.

78314 The overall *awk* syntax has always been based on the C language, with a few features from the  
 78315 shell command language and other sources. Because of this, it is not completely compatible with  
 78316 any other language, which has caused confusion for some users. It is not the intent of the  
 78317 standard developers to address such issues. A few relatively minor changes toward making the  
 78318 language more compatible with the ISO C standard were made; most of these changes are based  
 78319 on similar changes in recent implementations, as described above. There remain several C-  
 78320 language conventions that are not in *awk*. One of the notable ones is the `<comma>` operator,  
 78321 which is commonly used to specify multiple expressions in the C language `for` statement. Also,  
 78322 there are various places where *awk* is more restrictive than the C language regarding the type of  
 78323 expression that can be used in a given context. These limitations are due to the different features  
 78324 that the *awk* language does provide.

78325 Regular expressions in *awk* have been extended somewhat from historical implementations to  
 78326 make them a pure superset of extended regular expressions, as defined by POSIX.1-2008 (see  
 78327 XBD Section 9.4, on page 188). The main extensions are internationalization features and  
 78328 interval expressions. Historical implementations of *awk* have long supported  
 78329 `<backslash>`-escape sequences as an extension to extended regular expressions, and this  
 78330 extension has been retained despite inconsistency with other utilities. The number of escape  
 78331 sequences recognized in both extended regular expressions and strings has varied (generally  
 78332 increasing with time) among implementations. The set specified by POSIX.1-2008 includes most  
 78333 sequences known to be supported by popular implementations and by the ISO C standard. One  
 78334 sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would  
 78335 allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard.  
 78336 However, because this syntax has a non-deterministic length, it does not permit the subsequent  
 78337 character to be a hexadecimal digit. This limitation can be dealt with in the C language by the  
 78338 use of lexical string concatenation. In the *awk* language, concatenation could also be a solution  
 78339 for strings, but not for extended regular expressions (either lexical ERE tokens or strings used  
 78340 dynamically as regular expressions). Because of this limitation, the feature has not been added to  
 78341 POSIX.1-2008.

78342 When a string variable is used in a context where an extended regular expression normally  
 78343 appears (where the lexical token ERE is used in the grammar) the string does not contain the  
 78344 literal `<slash>` characters.

78345 Some versions of *awk* allow the form:

```
78346 func name(args, ... ) { statements }
```

78347 This has been deprecated by the authors of the language, who asked that it not be specified.

78348 Historical implementations of *awk* produce an error if a `next` statement is executed in a `BEGIN`  
 78349 action, and cause *awk* to terminate if a `next` statement is executed in an `END` action. This  
 78350 behavior has not been documented, and it was not believed that it was necessary to standardize  
 78351 it.

78352 The specification of conversions between string and numeric values is much more detailed than  
 78353 in the documentation of historical implementations or in the referenced *The AWK Programming*  
 78354 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to

78355 ensure compatible behavior from different implementations. This is especially important in  
 78356 relational expressions since the types of the operands determine whether a string or numeric  
 78357 comparison is performed. From the perspective of an application developer, it is usually  
 78358 sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating  
 78359 a null string) when the type of an expression does not obviously match what is needed. The  
 78360 intent has been to specify historical practice in almost all cases. The one exception is that, in  
 78361 historical implementations, variables and constants maintain both string and numeric values  
 78362 after their original value is converted by any use. This means that referencing a variable or  
 78363 constant can have unexpected side-effects. For example, with historical implementations the  
 78364 following program:

```
78365 {
78366     a = "+2"
78367     b = 2
78368     if (NR % 2)
78369         c = a + b
78370     if (a == b)
78371         print "numeric comparison"
78372     else
78373         print "string comparison"
78374 }
```

78375 would perform a numeric comparison (and output numeric comparison) for each odd-  
 78376 numbered line, but perform a string comparison (and output string comparison) for each even-  
 78377 numbered line. POSIX.1-2008 ensures that comparisons will be numeric if necessary. With  
 78378 historical implementations, the following program:

```
78379 BEGIN {
78380     OFMT = "%e"
78381     print 3.14
78382     OFMT = "%f"
78383     print 3.14
78384 }
```

78385 would output "3.140000e+00" twice, because in the second **print** statement the constant  
 78386 "3.14" would have a string value from the previous conversion. POSIX.1-2008 requires that the  
 78387 output of the second **print** statement be "3.140000". The behavior of historical  
 78388 implementations was seen as too unintuitive and unpredictable.

78389 It was pointed out that with the rules contained in early drafts, the following script would print  
 78390 nothing:

```
78391 BEGIN {
78392     y[1.5] = 1
78393     OFMT = "%e"
78394     print y[1.5]
78395 }
```

78396 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
 78397 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
 78398 conversions, such as comparisons or array indexing. The default value is the same as that for  
 78399 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
 78400 will receive the historical behavior associated with internal string conversions.

78401 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
 78402 sources. Again the intent has been to specify historical practice. One convention that may not be

78403 obvious from the formal grammar as in other verbal descriptions is where <newline> characters  
 78404 are acceptable. There are several obvious placements such as terminating a statement, and a  
 78405 <backslash> can be used to escape <newline> characters between any lexical tokens. In addition,  
 78406 <newline> characters without <backslash> characters can follow a comma, an open brace, a  
 78407 logical AND operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword,  
 78408 and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
78409 { print $1,  
78410     $2 }
```

78411 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify  
 78412 the grammar, making it match a text file in form. There is no way for an application or test suite  
 78413 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

78414 POSIX.1-2008 requires several changes from historical implementations in order to support  
 78415 internationalization. Probably the most subtle of these is the use of the decimal-point character,  
 78416 defined by the *LC\_NUMERIC* category of the locale, in representations of floating-point  
 78417 numbers. This locale-specific character is used in recognizing numeric input, in converting  
 78418 between strings and numeric values, and in formatting output. However, regardless of locale,  
 78419 the <period> character (the decimal-point character of the POSIX locale) is the decimal-point  
 78420 character recognized in processing *awk* programs (including assignments in command line  
 78421 arguments). This is essentially the same convention as the one used in the ISO C standard. The  
 78422 difference is that the C language includes the *setlocale()* function, which permits an application  
 78423 to modify its locale. Because of this capability, a C application begins executing with its locale set  
 78424 to the C locale, and only executes in the environment-specified locale after an explicit call to  
 78425 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as  
 78426 inappropriate for POSIX.1-2008. It is possible to execute an *awk* program explicitly in any desired  
 78427 locale by setting the environment in the shell.

78428 The undefined behavior resulting from NULs in extended regular expressions allows future  
 78429 extensions for the GNU *gawk* program to process binary data.

78430 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic  
 78431 errors) is undefined because it was considered overly limiting on implementations to specify. In  
 78432 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.  
 78433 However, some implementations may choose to extend the language in ways that make use of  
 78434 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but  
 78435 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect  
 78436 in some implementations. Also, different implementations might detect a given error during an  
 78437 initial parsing of the program (before reading any input files) while others might detect it when  
 78438 executing the program after reading some input. Implementors should be aware that diagnosing  
 78439 errors as early as possible and producing useful diagnostics can ease debugging of applications,  
 78440 and thus make an implementation more usable.

78441 The unspecified behavior from using multi-character **RS** values is to allow possible future  
 78442 extensions based on extended regular expressions used for record separators. Historical  
 78443 implementations take the first character of the string and ignore the others.

78444 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future  
 78445 extension that would split up a string into an array of individual characters.

78446 In the context of the **getline** function, equally good arguments for different precedences of the |  
 78447 and < operators can be made. Historical practice has been that:

```
78448 getline < "a" "b"
```

78449 is parsed as:

78450 ( getline < "a" ) "b"

78451 although many would argue that the intent was that the file **ab** should be read. However:

78452 getline < "x" + 1

78453 parses as:

78454 getline < ( "x" + 1 )

78455 Similar problems occur with the | version of **getline**, particularly in combination with **\$**. For  
78456 example:

78457 \$"echo hi" | getline

78458 (This situation is particularly problematic when used in a **print** statement, where the |**getline**  
78459 part might be a redirection of the **print**.)

78460 Since in most cases such constructs are not (or at least should not) be used (because they have a  
78461 natural ambiguity for which there is no conventional parsing), the meaning of these constructs  
78462 has been made explicitly unspecified. (The effect is that a conforming application that runs into  
78463 the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual  
78464 uses of such constructs.

78465 Grammars can be written that would cause an error under these circumstances. Where  
78466 backwards-compatibility is not a large consideration, implementors may wish to use such  
78467 grammars.

78468 Some historical implementations have allowed some built-in functions to be called without an  
78469 argument list, the result being a default argument list chosen in some "reasonable" way. Use of  
78470 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely  
78471 known or widely used; this particular form is documented in various places (for example, most  
78472 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as  
78473 legitimate practice. With this exception, default argument lists have always been undocumented  
78474 and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-  
78475 defined functions. They add no useful functionality and preclude possible future extensions that  
78476 might need to name functions without calling them. Not standardizing them seems the simplest  
78477 course. The standard developers considered that **length** merited special treatment, however,  
78478 since it has been documented in the past and sees possibly substantial use in historical  
78479 programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the  
78480 obsolescent marking for XSI-conforming implementations and many otherwise conforming  
78481 applications depend on this feature.

78482 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive  
78483 <backslash> characters should be used in the string to ensure a single <backslash> will precede  
78484 the <ampersand> when the resultant string is passed to the function. (For example, to specify  
78485 one literal <ampersand> in the replacement string, use **gsub(ERE, "\\&")**.)

78486 Historically, the only special character in the *repl* argument of **sub** and **gsub** string functions was  
78487 the <ampersand> ('&') character and preceding it with the <backslash> character was used to  
78488 turn off its special meaning.

78489 The description in the ISO POSIX-2:1993 standard introduced behavior such that the  
78490 <backslash> character was another special character and it was unspecified whether there were  
78491 any other special characters. This description introduced several portability problems, some of  
78492 which are described below, and so it has been replaced with the more historical description.  
78493 Some of the problems include:

- 78494 • Historically, to create the replacement string, a script could use `gsub(ERE, "\\&")`, but  
 78495 with the ISO POSIX-2:1993 standard wording, it was necessary to use `gsub(ERE,`  
 78496 `"\\\\&")`. The `<backslash>` characters are doubled here because all string literals are  
 78497 subject to lexical analysis, which would reduce each pair of `<backslash>` characters to a  
 78498 single `<backslash>` before being passed to `gsub`.
- 78499 • Since it was unspecified what the special characters were, for portable scripts to guarantee  
 78500 that characters are printed literally, each character had to be preceded with a `<backslash>`.  
 78501 (For example, a portable script had to use `gsub(ERE, "\\h\\i")` to produce a replacement  
 78502 string of "hi".)

78503 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe  
 78504 historical practice because of the way numeric strings are compared as numbers. The current  
 78505 rules cause the following code:

```
78506 if (0 == "000")
78507     print "strange, but true"
78508 else
78509     print "not true"
```

78510 to do a numeric comparison, causing the `if` to succeed. It should be intuitively obvious that this  
 78511 is incorrect behavior, and indeed, no historical implementation of `awk` actually behaves this way.

78512 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
 78513 obtained from specific circumstances (mostly external sources) where it is not possible to  
 78514 determine unambiguously whether the value is intended to be a string or a numeric.

78515 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
 78516 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
 78517 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
 78518 the numeric value zero.

78519 Uninitialized variables include all types of variables including scalars, array elements, and  
 78520 fields. The definition of an uninitialized value in [Variables and Special Variables](#) (on page 2437)  
 78521 is necessary to describe the value placed on uninitialized variables and on fields that are valid  
 78522 (for example, `< $NF`) but have no characters in them and to describe how these variables are to  
 78523 be used in comparisons. A valid field, such as `$1`, that has no characters in it can be obtained  
 78524 from an input line of `"\t\t\t"` when `FS='\t'`. Historically, the comparison (`$1<10`) was done  
 78525 numerically after evaluating `$1` to the value zero.

78526 The phrase `"... also shall have the numeric value of the numeric string"` was removed from  
 78527 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
 78528 implementation detail. It is not necessary for POSIX.1-2008 to specify that these objects be  
 78529 assigned two different values. It is only necessary to specify that these objects may evaluate to  
 78530 two different values depending on context.

78531 Historical implementations of `awk` did not parse hexadecimal integer or floating constants like  
 78532 `"0xa"` and `"0xap0"`. Due to an oversight, the 2001 through 2004 editions of this standard  
 78533 required support for hexadecimal floating constants. This was due to the reference to `atof()`.  
 78534 This version of the standard allows but does not require implementations to use `atof()` and  
 78535 includes a description of how floating-point numbers are recognized as an alternative to match  
 78536 historic behavior. The intent of this change is to allow implementations to recognize floating-  
 78537 point constants according to either the ISO/IEC 9899:1990 standard or ISO/IEC 9899:1999  
 78538 standard, and to allow (but not require) implementations to recognize hexadecimal integer  
 78539 constants.

78540 Historical implementations of `awk` did not support floating-point infinities and NaNs in *numeric*

78541 *strings*; e.g., "-INF" and "NaN". However, implementations that use the *atof()* or *strtod()*  
 78542 functions to do the conversion picked up support for these values if they used a  
 78543 ISO/IEC 9899:1999 standard version of the function instead of a ISO/IEC 9899:1990 standard  
 78544 version. Due to an oversight, the 2001 through 2004 editions of this standard did not allow  
 78545 support for infinities and NaNs, but in this revision support is allowed (but not required). This is  
 78546 a silent change to the behavior of *awk* programs; for example, in the POSIX locale the expression:

78547 (" -INF" + 0 < 0)

78548 formerly had the value 0 because "-INF" converted to 0, but now it may have the value 0 or 1.

#### 78549 FUTURE DIRECTIONS

78550 None.

#### 78551 SEE ALSO

78552 [Section 1.3](#) (on page 2287), *grep*, *lex*, *sed*

78553 XBD [Chapter 5](#) (on page 121), [Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Chapter 9](#) (on  
 78554 page 181), [Section 12.2](#) (on page 215)

78555 XSH *atof()*, *exec*, *isspace()*, *popen()*, *setlocale()*, *strtod()*

#### 78556 CHANGE HISTORY

78557 First released in Issue 2.

#### 78558 Issue 5

78559 The FUTURE DIRECTIONS section is added.

#### 78560 Issue 6

78561 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

78562 The normative text is reworded to avoid use of the term "must" for application requirements.

78563 IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence "An occurrence of two  
 78564 consecutive <backslash> characters shall be interpreted as just a single literal <backslash>  
 78565 character." into the description of the **sub** string function.

#### 78566 Issue 7

78567 PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of  
 78568 the **OFS** variable.

78569 Austin Group Interpretation 1003.1-2001 #189 is applied.

78570 Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support  
 78571 infinities and NaNs.

78572 SD5-XCU-ERN-79 is applied, restoring the horizontal lines to [Table 4-1](#) (on page 2433), and  
 78573 SD5-XCU-ERN-80 is applied, changing the order of some table entries.

78574 SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.

78575 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

78576 The EXTENDED DESCRIPTION is changed to make the support of hexadecimal integer and  
 78577 floating constants optional.

**basename**

Utilities

78578 **NAME**78579 `basename` — return non-directory portion of a pathname78580 **SYNOPSIS**78581 `basename string [suffix]`78582 **DESCRIPTION**

78583 The *string* operand shall be treated as a pathname, as defined in XBD Section 3.266 (on page 75).  
 78584 The string *string* shall be converted to the filename corresponding to the last pathname  
 78585 component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be  
 78586 done by performing actions equivalent to the following steps in order:

- 78587 1. If *string* is a null string, it is unspecified whether the resulting string is `./` or a null  
 78588 string. In either case, skip steps 2 through 6.
- 78589 2. If *string* is `"/"`, it is implementation-defined whether steps 3 to 6 are skipped or  
 78590 processed.
- 78591 3. If *string* consists entirely of `<slash>` characters, *string* shall be set to a single `<slash>`  
 78592 character. In this case, skip steps 4 to 6.
- 78593 4. If there are any trailing `<slash>` characters in *string*, they shall be removed.
- 78594 5. If there are any `<slash>` characters remaining in *string*, the prefix of *string* up to and  
 78595 including the last `<slash>` character in *string* shall be removed.
- 78596 6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 78597 identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 78598 from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an  
 78599 error if *suffix* is not found in *string*.

78600 The resulting string shall be written to standard output.

78601 **OPTIONS**

78602 None.

78603 **OPERANDS**

78604 The following operands shall be supported:

78605 *string* A string.

78606 *suffix* A string.

78607 **STDIN**

78608 Not used.

78609 **INPUT FILES**

78610 None.

78611 **ENVIRONMENT VARIABLES**78612 The following environment variables shall affect the execution of *basename*:

78613 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 78614 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 78615 variables used to determine the values of locale categories.)

78616 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 78617 internationalization variables.

78618 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 78619 characters (for example, single-byte as opposed to multi-byte characters in  
 78620 arguments).

78621 *LC\_MESSAGES*

78622 Determine the locale that should be used to affect the format and contents of

78623 diagnostic messages written to standard error.

78624 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

78625 **ASYNCHRONOUS EVENTS**

78626 Default.

78627 **STDOUT**

78628 The *basename* utility shall write a line to the standard output in the following format:

78629 "%s\n", <resulting string>

78630 **STDERR**

78631 The standard error shall be used only for diagnostic messages.

78632 **OUTPUT FILES**

78633 None.

78634 **EXTENDED DESCRIPTION**

78635 None.

78636 **EXIT STATUS**

78637 The following exit values shall be returned:

78638 0 Successful completion.

78639 >0 An error occurred.

78640 **CONSEQUENCES OF ERRORS**

78641 Default.

78642 **APPLICATION USAGE**

78643 The definition of *pathname* specifies implementation-defined behavior for pathnames starting

78644 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters

78645 to the beginning of a pathname unless they can ensure that there are more or less than two or are

78646 prepared to deal with the implementation-defined consequences.

78647 **EXAMPLES**

78648 If the string *string* is a valid pathname:

78649 `$(basename "string")`

78650 produces a filename that could be used to open the file named by *string* in the directory returned

78651 by:

78652 `$(dirname "string")`

78653 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be

78654 a valid filename. The *basename* utility is not expected to make any judgements about the validity

78655 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

78656 The following shell script compiles `/usr/src/cmd/cat.c` and moves the output to a file named `cat`

78657 in the current directory when invoked with the argument `/usr/src/cmd/cat` or with the argument

78658 `/usr/src/cmd/cat.c`:

78659 `c99 $(dirname "$1")/$(basename "$1" .c) .c`

78660 `mv a.out $(basename "$1" .c)`

78661 **RATIONALE**

78662 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid  
78663 pathname:

78664 `$(basename "string")`

78665 would be a valid filename for the file in the directory:

78666 `$(dirname "string")`

78667 This would not work for the early proposal versions of these utilities due to the way it specified  
78668 handling of trailing <slash> characters.

78669 Since the definition of *pathname* specifies implementation-defined behavior for pathnames  
78670 starting with two <slash> characters, this volume of POSIX.1-2008 specifies similar  
78671 implementation-defined behavior for the *basename* and *dirname* utilities.

78672 **FUTURE DIRECTIONS**

78673 None.

78674 **SEE ALSO**

78675 [Section 2.5](#) (on page 2301), *dirname*

78676 XBD [Section 3.266](#) (on page 75), [Chapter 8](#) (on page 173)

78677 **CHANGE HISTORY**

78678 First released in Issue 2.

78679 **Issue 6**

78680 IEEE PASC Interpretation 1003.2 #164 is applied.

78681 The normative text is reworded to avoid use of the term "must" for application requirements.

78682 **NAME**

78683 batch — schedule commands to be executed in a batch queue

78684 **SYNOPSIS**78685 *batch*78686 **DESCRIPTION**78687 The *batch* utility shall read commands from standard input and schedule them for execution in a  
78688 batch queue. It shall be the equivalent of the command:78689 `at -q b -m now`78690 where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to  
78691 the batch queue with no time constraints and shall be run by the system using algorithms, based  
78692 on unspecified factors, that may vary with each invocation of *batch*.78693 XSI Users shall be permitted to use *batch* if their name appears in the file **at.allow**, which is located in  
78694 an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located  
78695 in an implementation-defined directory, shall be checked to determine whether the user shall be  
78696 denied access to *batch*. If neither file exists, only a process with appropriate privileges shall be  
78697 allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The  
78698 **at.allow** and **at.deny** files shall consist of one user name per line.78699 **OPTIONS**

78700 None.

78701 **OPERANDS**

78702 None.

78703 **STDIN**78704 The standard input shall be a text file consisting of commands acceptable to the shell command  
78705 language described in Chapter 2 (on page 2297).78706 **INPUT FILES**78707 XSI The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory,  
78708 shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
78709 denied access to the *at* and *batch* utilities.78710 **ENVIRONMENT VARIABLES**78711 The following environment variables shall affect the execution of *batch*:78712 **LANG** Provide a default value for the internationalization variables that are unset or null.  
78713 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
78714 variables used to determine the values of locale categories.)78715 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
78716 internationalization variables.78717 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
78718 characters (for example, single-byte as opposed to multi-byte characters in  
78719 arguments and input files).78720 **LC\_MESSAGES**78721 Determine the locale that should be used to affect the format and contents of  
78722 diagnostic messages written to standard error and informative messages written to  
78723 standard output.78724 **LC\_TIME** Determine the format and contents for date and time strings written by *batch*.

78725	XSI	<b>NLSPATH</b>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
78726		<b>SHELL</b>	Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen.
78727			
78728			
78729			
78730			
78731		<b>TZ</b>	Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
78732			
78733			
78734			
78735			
78736		<b>ASYNCHRONOUS EVENTS</b>	
78737		Default.	
78738		<b>STDOUT</b>	
78739			When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.
78740			
78741		<b>STDERR</b>	
78742			The following shall be written to standard error when a job has been successfully submitted:
78743			"job %s at %s\n", <i>at_job_id</i> , <date>
78744			where <i>date</i> shall be equivalent in format to the output of:
78745			date +"%a %b %e %T %Y"
78746			The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).
78747			
78748			Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.
78749			
78750			Diagnostic messages, if any, shall be written to standard error.
78751		<b>OUTPUT FILES</b>	
78752		None.	
78753		<b>EXTENDED DESCRIPTION</b>	
78754		None.	
78755		<b>EXIT STATUS</b>	
78756			The following exit values shall be returned:
78757		0	Successful completion.
78758		>0	An error occurred.
78759		<b>CONSEQUENCES OF ERRORS</b>	
78760			The job shall not be scheduled.

78761 **APPLICATION USAGE**

78762 It may be useful to redirect standard output within the specified commands.

78763 **EXAMPLES**

78764 1. This sequence can be used at a terminal:

78765 

```
batch
78766 sort < file >outfile
78767 EOT
```

78768 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
78769 command procedure (the sequence of output redirection specifications is significant):78770 

```
batch <<!
78771 diff file1 file2 2>&1 >outfile | mailx mygroup
78772 !
```

78773 **RATIONALE**78774 Early proposals described *batch* in a manner totally separated from *at*, even though the historical  
78775 model treated it almost as a synonym for *at -qb*. A number of features were added to list and  
78776 control batch work separately from those in *at*. Upon further reflection, it was decided that the  
78777 benefit of this did not merit the change to the historical interface.78778 The *-m* option was included on the equivalent *at* command because it is historical practice to  
78779 mail results to the submitter, even if all job-produced output is redirected. As explained in the  
78780 RATIONALE for *at*, the *now* keyword submits the job for immediate execution (after scheduling  
78781 delays), despite some historical systems where *at now* would have been considered an error.78782 **FUTURE DIRECTIONS**

78783 None.

78784 **SEE ALSO**78785 *at*

78786 XBD Chapter 8 (on page 173)

78787 **CHANGE HISTORY**

78788 First released in Issue 2.

78789 **Issue 6**

78790 This utility is marked as part of the User Portability Utilities option.

78791 The NAME is changed to align with the IEEE P1003.2b draft standard.

78792 The normative text is reworded to avoid use of the term “must” for application requirements.

78793 **Issue 7**78794 The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability  
78795 Utilities is now an option for interactive utilities.78796 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
78797 by the *batch* utility.

78798 **NAME**

78799 bc — arbitrary-precision arithmetic language

78800 **SYNOPSIS**78801 bc [-l] [*file...*]78802 **DESCRIPTION**

78803 The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files  
 78804 given, then read from the standard input. If the standard input and standard output to *bc* are  
 78805 attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing  
 78806 behavioral constraints described in the following sections.

78807 **OPTIONS**78808 The *bc* utility shall conform to XBD Section 12.2 (on page 215).

78809 The following option shall be supported:

78810 **-l** (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 78811 default zero; see the EXTENDED DESCRIPTION section.

78812 **OPERANDS**

78813 The following operand shall be supported:

78814 *file* A pathname of a text file containing *bc* program statements. After all *files* have  
 78815 been read, *bc* shall read the standard input.

78816 **STDIN**

78817 See the INPUT FILES section.

78818 **INPUT FILES**

78819 Input files shall be text files containing a sequence of comments, statements, and function  
 78820 definitions that shall be executed as they are read.

78821 **ENVIRONMENT VARIABLES**78822 The following environment variables shall affect the execution of *bc*:

78823 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 78824 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 78825 variables used to determine the values of locale categories.)

78826 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 78827 internationalization variables.

78828 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 78829 characters (for example, single-byte as opposed to multi-byte characters in  
 78830 arguments and input files).

78831 **LC\_MESSAGES**

78832 Determine the locale that should be used to affect the format and contents of  
 78833 diagnostic messages written to standard error.

78834 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

78835 **ASYNCHRONOUS EVENTS**

78836 Default.

78837 **STDOUT**

78838 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 78839 lines containing the value of all executed expressions without assignments. The radix and  
 78840 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 78841 EXTENDED DESCRIPTION section.

78842 **STDERR**

78843 The standard error shall be used only for diagnostic messages.

78844 **OUTPUT FILES**

78845 None.

78846 **EXTENDED DESCRIPTION**78847 **Grammar**

78848 The grammar in this section and the lexical conventions in the following section shall together  
 78849 describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 78850 described in [Section 1.3](#) (on page 2287). A valid program can be represented as the non-terminal  
 78851 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax  
 78852 description.

```

78853 %token      EOF NEWLINE STRING LETTER NUMBER
78854 %token      MUL_OP
78855 /*         '*' , '/' , '%' */
78856 %token      ASSIGN_OP
78857 /*         '=' , '+=' , '--=' , '*=' , '/=' , '%=' , '^=' */
78858 %token      REL_OP
78859 /*         '==' , '<=' , '>=' , '!=' , '<' , '>' */
78860 %token      INCR_DECR
78861 /*         '++' , '--' */
78862 %token      Define      Break      Quit      Length
78863 /*         'define' , 'break' , 'quit' , 'length' */
78864 %token      Return      For      If      While      Sqrt
78865 /*         'return' , 'for' , 'if' , 'while' , 'sqrt' */
78866 %token      Scale      Ibase      Obase      Auto
78867 /*         'scale' , 'ibase' , 'obase' , 'auto' */
78868 %start      program
78869 %%
78870 program      : EOF
78871              | input_item program
78872              ;
78873 input_item   : semicolon_list NEWLINE
78874              | function
78875              ;
78876 semicolon_list : /* empty */
78877              | statement
78878              | semicolon_list ';' statement
78879              | semicolon_list ';'
78880              ;
78881 statement_list : /* empty */
78882              | statement
78883              | statement_list NEWLINE

```

```

78884 | statement_list NEWLINE statement
78885 | statement_list ';'
78886 | statement_list ';' statement
78887 ;
78888 statement : expression
78889 | STRING
78890 | Break
78891 | Quit
78892 | Return
78893 | Return '(' return_expression ')'
78894 | For '(' expression ';'
78895 | relational_expression ';'
78896 | expression ')' statement
78897 | If '(' relational_expression ')' statement
78898 | While '(' relational_expression ')' statement
78899 | '{' statement_list '}'
78900 ;
78901 function : Define LETTER '(' opt_parameter_list ')'
78902 | '{' NEWLINE opt_auto_define_list
78903 | statement_list '}'
78904 ;
78905 opt_parameter_list : /* empty */
78906 | parameter_list
78907 ;
78908 parameter_list : LETTER
78909 | define_list ',' LETTER
78910 ;
78911 opt_auto_define_list : /* empty */
78912 | Auto define_list NEWLINE
78913 | Auto define_list ';'
78914 ;
78915 define_list : LETTER
78916 | LETTER '[' ']'
78917 | define_list ',' LETTER
78918 | define_list ',' LETTER '[' ']'
78919 ;
78920 opt_argument_list : /* empty */
78921 | argument_list
78922 ;
78923 argument_list : expression
78924 | LETTER '[' ']' ',' argument_list
78925 ;
78926 relational_expression : expression
78927 | expression REL_OP expression
78928 ;
78929 return_expression : /* empty */
78930 | expression

```

```

78931                                     ;
78932     expression                       : named_expression
78933                                       | NUMBER
78934                                       | '(' expression ')'
78935                                       | LETTER '(' opt_argument_list ')'
78936                                       | '-' expression
78937                                       | expression '+' expression
78938                                       | expression '-' expression
78939                                       | expression MUL_OP expression
78940                                       | expression '^' expression
78941                                       | INCR_DECR named_expression
78942                                       | named_expression INCR_DECR
78943                                       | named_expression ASSIGN_OP expression
78944                                       | Length '(' expression ')'
78945                                       | Sqrt '(' expression ')'
78946                                       | Scale '(' expression ')'
78947                                     ;
78948     named_expression                   : LETTER
78949                                       | LETTER '[' expression ']'
78950                                       | Scale
78951                                       | Ibase
78952                                       | Obase
78953                                     ;

```

### 78954 Lexical Conventions in bc

78955 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
78956 follows:

- 78957 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
78958 given point.
- 78959 2. A comment shall consist of any characters beginning with the two adjacent characters  
78960 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
78961 Comments shall have no effect except to delimit lexical tokens.
- 78962 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 78963 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
78964 beginning with the double-quote character (`'"`) and terminated by another occurrence  
78965 of the double-quote character. The value of the string is the sequence of all characters  
78966 between, but not including, the two double-quote characters. All characters shall be taken  
78967 literally from the input, and there is no way to specify a string containing a double-quote  
78968 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`  
78969 bytes.
- 78970 5. A `<blank>` shall have no effect except as an ordinary character if it appears within a  
78971 **STRING** token, or to delimit a lexical token other than **STRING**.
- 78972 6. The combination of a `<backslash>` character immediately followed by a `<newline>` shall  
78973 have no effect other than to delimit lexical tokens with the following exceptions:

- 78974                   • It shall be interpreted as the character sequence "`\<newline>`" in **STRING** tokens.
- 78975                   • It shall be ignored as part of a multi-line **NUMBER** token.
- 78976           7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the  
78977 following grammar:
- 78978           NUMBER : integer  
78979                   | '.' integer  
78980                   | integer '.'  
78981                   | integer '.' integer  
78982                   ;
- 78983           integer : digit  
78984                   | integer digit  
78985                   ;
- 78986           digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7  
78987                   | 8 | 9 | A | B | C | D | E | F  
78988                   ;
- 78989           8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by  
78990 the value of the internal register **ibase** (described below). Each of the **digit** characters  
78991 shall have the value from 0 to 15 in the order listed here, and the `<period>` character shall  
78992 represent the radix point. The behavior is undefined if digits greater than or equal to the  
78993 value of **ibase** appear in the token. However, note the exception for single-digit values  
78994 being assigned to **ibase** and **obase** themselves, in [Operations in bc](#) (on page 2475).
- 78995           9. The following keywords shall be recognized as tokens:
- 78996           **auto**       **ibase**       **length**   **return**   **while**  
78997           **break**   **if**       **obase**   **scale**  
78998           **define**   **for**      **quit**   **sqrt**
- 78999           10. Any of the following characters occurring anywhere except within a keyword shall be  
79000 recognized as the token **LETTER**:
- 79001           a b c d e f g h i j k l m n o p q r s t u v w x y z
- 79002           11. The following single-character and two-character sequences shall be recognized as the  
79003 token **ASSIGN\_OP**:
- 79004           = += -= \*= /= %= ^=
- 79005           12. If an '=' character, as the beginning of a token, is followed by a '-' character with no  
79006 intervening delimiter, the behavior is undefined.
- 79007           13. The following single-characters shall be recognized as the token **MUL\_OP**:
- 79008           \* / %
- 79009           14. The following single-character and two-character sequences shall be recognized as the  
79010 token **REL\_OP**:
- 79011           == <= >= != < >
- 79012           15. The following two-character sequences shall be recognized as the token **INCR\_DECR**:
- 79013           ++ --

79014 16. The following single characters shall be recognized as tokens whose names are the  
79015 character:

79016 <newline> ( ) , + - ; [ ] ^ { }

79017 17. The token **EOF** is returned when the end of input is reached.

### 79018 Operations in bc

79019 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function  
79020 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed  
79021 by square brackets ("[]"). An array subscript is required except in an argument or auto list.  
79022 Arrays are singly dimensioned and can contain up to {BC\_DIM\_MAX} elements. Indexing shall  
79023 begin at zero so an array is indexed from 0 to {BC\_DIM\_MAX}-1. Subscripts shall be truncated  
79024 to integers. The application shall ensure that function identifiers are followed by parentheses,  
79025 possibly enclosing arguments. The three types of identifiers do not conflict.

79026 The following table summarizes the rules for precedence and associativity of all operators.  
79027 Operators on the same line shall have the same precedence; rows are in order of decreasing  
79028 precedence.

79029 **Table 4-3** Operators in *bc*

Operator	Associativity
++, --	N/A
unary -	N/A
^	Right to left
*, /, %	Left to right
+, binary -	Left to right
=, +=, -=, *=, /=, %=, ^=	Right to left
==, <=, >=, !=, <, >	None

79038 Each expression or named expression has a *scale*, which is the number of decimal digits that  
79039 shall be maintained as the fractional portion of the expression.

79040 *Named expressions* are places where values are stored. Named expressions shall be valid on the  
79041 left side of an assignment. The value of a named expression shall be the value stored in the place  
79042 named. Simple identifiers and array elements are named expressions; they have an initial value  
79043 of zero and an initial scale of zero.

79044 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an  
79045 expression consisting of the name of one of these registers shall be zero; values assigned to any  
79046 of these registers are truncated to integers. The **scale** register shall contain a global value used in  
79047 computing the scale of expressions (as described below). The value of the register **scale** is  
79048 limited to  $0 \leq \text{scale} \leq \{\text{BC\_SCALE\_MAX}\}$  and shall have a default value of zero. The **ibase** and  
79049 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be  
79050 limited to:

79051  $2 \leq \text{ibase} \leq 16$

79052 The value of **obase** shall be limited to:

79053  $2 \leq \text{obase} \leq \{\text{BC\_BASE\_MAX}\}$

79054 When either **ibase** or **obase** is assigned a single **digit** value from the list in [Lexical Conventions](#)  
79055 [in bc](#) (on page 2473), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to  
79056 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when

79057 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall  
79058 have initial values of 10.

79059 Internal computations shall be conducted as if in decimal, regardless of the input and output  
79060 bases, to the specified number of decimal digits. When an exact result is not achieved (for  
79061 example, **scale**=0; 3.2/1), the result shall be truncated.

79062 For all values of **obase** specified by this volume of POSIX.1-2008, *bc* shall output numeric values  
79063 by performing each of the following steps in order:

- 79064 1. If the value is less than zero, a <hyphen> ( ' - ' ) character shall be output.
- 79065 2. One of the following is output, depending on the numerical value:
  - 79066 • If the absolute value of the numerical value is greater than or equal to one, the  
79067 integer portion of the value shall be output as a series of digits appropriate to **obase**  
79068 (as described below), most significant digit first. The most significant non-zero digit  
79069 shall be output next, followed by each successively less significant digit.
  - 79070 • If the absolute value of the numerical value is less than one but greater than zero  
79071 and the scale of the numerical value is greater than zero, it is unspecified whether  
79072 the character 0 is output.
  - 79073 • If the numerical value is zero, the character 0 shall be output.
- 79074 3. If the scale of the value is greater than zero and the numeric value is not zero, a <period>  
79075 character shall be output, followed by a series of digits appropriate to **obase** (as described  
79076 below) representing the most significant portion of the fractional part of the value. If *s*  
79077 represents the scale of the value being output, the number of digits output shall be *s* if  
79078 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s*  
79079 if **obase** is less than 10. For **obase** values other than 10, this should be the number of  
79080 digits needed to represent a precision of  $10^s$ .

79081 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

79082 0 1 2 3 4 5 6 7 8 9 A B C D E F

79083 which represent the values zero to 15, inclusive, respectively.

79084 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.  
79085 Each digit except the most significant fractional digit shall be preceded by a single <space>. For  
79086 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-  
79087 digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be  
79088 written as:

79089 Δ01Δ15Δ24

79090 and in base 125, as:

79091 Δ008Δ024

79092 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;  
79093 other locales may split at different character boundaries. Lines that are continued shall end with  
79094 a <backslash>.

79095 A function call shall consist of a function name followed by parentheses containing a  
79096 <comma>-separated list of expressions, which are the function arguments. A whole array  
79097 passed as an argument shall be specified by the array name followed by empty square brackets.  
79098 All function arguments shall be passed by value. As a result, changes made to the formal  
79099 parameters shall have no effect on the actual arguments. If the function terminates by executing

- 79100 a **return** statement, the value of the function shall be the value of the expression in the  
 79101 parentheses of the **return** statement or shall be zero if no expression is provided or if there is no  
 79102 **return** statement.
- 79103 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be  
 79104 truncated in the least significant decimal place. The scale of the result shall be the scale of the  
 79105 expression or the value of **scale**, whichever is larger.
- 79106 The result of **length**(*expression*) shall be the total number of significant decimal digits in the  
 79107 expression. The scale of the result shall be zero.
- 79108 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be  
 79109 zero.
- 79110 A numeric constant shall be an expression. The scale shall be the number of digits that follow the  
 79111 radix point in the input representing the constant, or zero if no radix point appears.
- 79112 The sequence ( *expression* ) shall be an expression with the same value and scale as *expression*.  
 79113 The parentheses can be used to alter the normal precedence.
- 79114 The semantics of the unary and binary operators are as follows:
- 79115 *-expression*  
 79116 The result shall be the negative of the *expression*. The scale of the result shall be the scale of  
 79117 *expression*.
- 79118 The unary increment and decrement operators shall not modify the scale of the named  
 79119 expression upon which they operate. The scale of the result shall be the scale of that named  
 79120 expression.
- 79121 *++named-expression*  
 79122 The named expression shall be incremented by one. The result shall be the value of the  
 79123 named expression after incrementing.
- 79124 *--named-expression*  
 79125 The named expression shall be decremented by one. The result shall be the value of the  
 79126 named expression after decrementing.
- 79127 *named-expression++*  
 79128 The named expression shall be incremented by one. The result shall be the value of the  
 79129 named expression before incrementing.
- 79130 *named-expression--*  
 79131 The named expression shall be decremented by one. The result shall be the value of the  
 79132 named expression before decrementing.
- 79133 The exponentiation operator, <circumflex> ( ' ^ ' ), shall bind right to left.
- 79134 *expression^expression*  
 79135 The result shall be the first *expression* raised to the power of the second *expression*. If the  
 79136 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left  
 79137 expression and *b* is the absolute value of the right expression, the scale of the result shall be:  
 79138 if  $b \geq 0$   $\min(a * b, \max(\text{scale}, a))$  if  $b < 0$  *scale*
- 79139 The multiplicative operators ( ' \* ' , ' / ' , ' % ' ) shall bind left to right.
- 79140 *expression\*expression*  
 79141 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two  
 79142 expressions, then the scale of the result shall be:

79143  $\min(a+b, \max(\text{scale}, a, b))$

79144 *expression*/*expression*

79145 The result shall be the quotient of the two expressions. The scale of the result shall be the  
79146 value of **scale**.

79147 *expression*%*expression*

79148 For expressions *a* and *b*, *a*%*b* shall be evaluated equivalent to the steps:

79149 1. Compute *a*/*b* to current scale.

79150 2. Use the result to compute:

79151  $a - (a / b) * b$

79152 to scale:

79153  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

79154 The scale of the result shall be:

79155  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

79156 When **scale** is zero, the '**%**' operator is the mathematical remainder operator.

79157 The additive operators ('+', '-') shall bind left to right.

79158 *expression*+*expression*

79159 The result shall be the sum of the two expressions. The scale of the result shall be the  
79160 maximum of the scales of the expressions.

79161 *expression*-*expression*

79162 The result shall be the difference of the two expressions. The scale of the result shall be the  
79163 maximum of the scales of the expressions.

79164 The assignment operators ('=', '+=', '-=', '\*=', '/=', '%=', '^=') shall bind right to left.

79165 *named-expression*=*expression*

79166 This expression shall result in assigning the value of the expression on the right to the  
79167 named expression on the left. The scale of both the named expression and the result shall be  
79168 the scale of *expression*.

79169 The compound assignment forms:

79170 *named-expression* <operator>= *expression*

79171 shall be equivalent to:

79172 *named-expression*=*named-expression* <operator> *expression*

79173 except that the *named-expression* shall be evaluated only once.

79174 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be  
79175 only valid as the object of an **if**, **while**, or inside a **for** statement.

79176 *expression1*<*expression2*

79177 The relation shall be true if the value of *expression1* is strictly less than the value of  
79178 *expression2*.

79179 *expression1*>*expression2*

79180 The relation shall be true if the value of *expression1* is strictly greater than the value of  
79181 *expression2*.

79182 *expression1* <= *expression2*

79183 The relation shall be true if the value of *expression1* is less than or equal to the value of  
79184 *expression2*.

79185 *expression1* >= *expression2*

79186 The relation shall be true if the value of *expression1* is greater than or equal to the value of  
79187 *expression2*.

79188 *expression1* = *expression2*

79189 The relation shall be true if the values of *expression1* and *expression2* are equal.

79190 *expression1* != *expression2*

79191 The relation shall be true if the values of *expression1* and *expression2* are unequal.

79192 There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are  
79193 local to a function need be declared with the **auto** command. The arguments to a function shall  
79194 be local to the function. All other identifiers are assumed to be global and available to all  
79195 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto  
79196 shall be allocated on entry to the function and released on returning from the function. They  
79197 therefore do not retain values between function calls. Auto arrays shall be specified by the array  
79198 name followed by empty square brackets. On entry to a function, the old values of the names  
79199 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the  
79200 function returns, reference to these names shall refer only to the new values.

79201 References to any of these names from other functions that are called from this function also  
79202 refer to the new value until one of those functions uses the same name for a local variable.

79203 When a statement is an expression, unless the main operator is an assignment, execution of the  
79204 statement shall write the value of the expression followed by a <newline>.

79205 When a statement is a string, execution of the statement shall write the value of the string.

79206 Statements separated by <semicolon> or <newline> characters shall be executed sequentially. In  
79207 an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical  
79208 production:

79209 `input_item : semicolon_list NEWLINE`

79210 the sequential list of statements making up the **semicolon\_list** shall be executed immediately  
79211 and any output produced by that execution shall be written without any delay due to buffering.

79212 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

79213 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;  
79214 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the  
79215 *relation* is false, execution shall resume after *statement*.

79216 A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

79217 `first-expression`  
79218 `while (relation) {`  
79219 `statement`  
79220 `last-expression`  
79221 `}`

79222 The application shall ensure that all three expressions are present.

79223 The **break** statement shall cause termination of a **for** or **while** statement.

79224 The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be

79225 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers  
79226 shall be specified by following the array name by empty square brackets. The application shall  
79227 ensure that the **auto** statement is the first statement in a function definition.

79228 A **define** statement:

```
79229 define LETTER ( opt_parameter_list ) {
79230     opt_auto_define_list
79231     statement_list
79232 }
```

79233 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the  
79234 **define** statement shall replace the previous definition. The expression:

```
79235 LETTER ( opt_argument_list )
```

79236 shall invoke the function named **LETTER**. The behavior is undefined if the number of  
79237 arguments in the invocation does not match the number of parameters in the definition.  
79238 Functions shall be defined before they are invoked. A function shall be considered to be defined  
79239 within its own body, so recursive calls are valid. The values of numeric constants within a  
79240 function shall be interpreted in the base specified by the value of the **ibase** register when the  
79241 function is invoked.

79242 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,  
79243 popping of its auto variables, and specification of the result of the function. The first form shall  
79244 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the  
79245 value and scale of the expression returned.

79246 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement  
79247 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

79248 The following functions shall be defined when the **-I** option is specified:

```
79249 s( expression )
79250     Sine of argument in radians.
79251 c( expression )
79252     Cosine of argument in radians.
79253 a( expression )
79254     Arctangent of argument.
79255 l( expression )
79256     Natural logarithm of argument.
79257 e( expression )
79258     Exponential function of argument.
79259 j( expression, expression )
79260     Bessel function of integer order.
```

79261 The scale of the result returned by these functions shall be the value of the **scale** register at the  
79262 time the function is invoked. The value of the **scale** register after these functions have completed  
79263 their execution shall be the same value it had upon invocation. The behavior is undefined if any  
79264 of these functions is invoked with an argument outside the domain of the mathematical  
79265 function.

79266 **EXIT STATUS**

79267 The following exit values shall be returned:

79268 0 All input files were processed successfully.

79269 *unspecified* An error occurred.79270 **CONSEQUENCES OF ERRORS**79271 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.79272  
79273 In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.79276 **APPLICATION USAGE**79277 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.79278 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.79282 The *bc* utility always uses the <period> ( ' . ' ) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the <period> character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a <comma> as the decimal-point character:79289 

```
define f(a,b) {
79290     ...
79291 }
79292 ...
79293 f(1,2,3)
```

79294 Because of such ambiguities, the &lt;period&gt; character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the &lt;period&gt; is also used in output.

79297 **EXAMPLES**79298 In the shell, the following assigns an approximation of the first ten digits of ' $\pi$ ' to the variable *x*:79299 

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

79300 The following *bc* program prints the same approximation of ' $\pi$ ', with a label, to standard output:79302 

```
scale = 10
79303 "pi equals "
79304 104348 / 33215
```

79305 The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the *-l* option is specified):79307 

```
scale = 20
79308 define e(x) {
79309     auto a, b, c, i, s
```

```

79310         a = 1
79311         b = 1
79312         s = 1
79313         for (i = 1; 1 == 1; i++){
79314             a = a*x
79315             b = b*i
79316             c = a/b
79317             if (c == 0) {
79318                 return(s)
79319             }
79320             s = s+c
79321         }
79322     }

```

79323 The following prints approximate values of the exponential function of the first ten integers:

```

79324     for (i = 1; i <= 10; ++i) {
79325         e(i)
79326     }

```

#### 79327 RATIONALE

79328 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to  
79329 be part of this volume of POSIX.1-2008 because *bc* was thought to have a more intuitive  
79330 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be  
79331 compliant.

79332 The exit status for error conditions has been left unspecified for several reasons:

- 79333 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes  
79334 may be appropriate for the two uses.
- 79335 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions,  
79336 and syntax errors are all possibilities.
- 79337 • It is not clear what utility the exit status has.
- 79338 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with  
79339 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*  
79340 aborted.

79341 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief  
79342 that *bc file1 file2* is used most often when at least *file1* contains data/function  
79343 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not  
79344 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check  
79345 all its files for accessibility before opening any of them.

79346 There was considerable debate on the appropriateness of the language accepted by *bc*. Several  
79347 reviewers preferred to see either a pure subset of the C language or some changes to make the  
79348 language more compatible with C. While the *bc* language has some obvious similarities to C, it  
79349 has never claimed to be compatible with any version of C. An interpreter for a subset of C might  
79350 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility  
79351 is known in historical practice, and it was not within the scope of this volume of POSIX.1-2008 to  
79352 define such a language and utility. If and when they are defined, it may be appropriate to  
79353 include them in a future version of this standard. This left the following alternatives:

- 79354 1. Exclude any calculator language from this volume of POSIX.1-2008.
- 79355 The consensus of the standard developers was that a simple programmatic calculator  
79356 language is very useful for both applications and interactive users. The only arguments  
79357 for excluding any calculator were that it would become obsolete if and when a C-  
79358 compatible one emerged, or that the absence would encourage the development of such a  
79359 C-compatible one. These arguments did not sufficiently address the needs of current  
79360 application developers.
- 79361 2. Standardize the historical *dc*, possibly with minor modifications.
- 79362 The consensus of the standard developers was that *dc* is a fundamentally less usable  
79363 language and that that would be far too severe a penalty for avoiding the issue of being  
79364 similar to but incompatible with C.
- 79365 3. Standardize the historical *bc*, possibly with minor modifications.
- 79366 This was the approach taken. Most of the proponents of changing the language would not  
79367 have been satisfied until most or all of the incompatibilities with C were resolved. Since  
79368 most of the changes considered most desirable would break historical applications and  
79369 require significant modification to historical implementations, almost no modifications  
79370 were made. The one significant modification that was made was the replacement of the  
79371 historical *bc* assignment operators "*=*", and so on, with the more modern "*+=*", and so  
79372 on. The older versions are considered to be fundamentally flawed because of the lexical  
79373 ambiguity in uses like *a=-1*.
- 79374 In order to permit implementations to deal with backwards-compatibility as they see fit,  
79375 the behavior of this one ambiguous construct was made undefined. (At least three  
79376 implementations have been known to support this change already, so the degree of  
79377 change involved should not be great.)
- 79378 The '*%*' operator is the mathematical remainder operator when **scale** is zero. The behavior of  
79379 this operator for other values of **scale** is from historical implementations of *bc*, and has been  
79380 maintained for the sake of historical applications despite its non-intuitive nature.
- 79381 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This  
79382 includes values less than 2, which were not seen as sufficiently useful to standardize. These  
79383 implementations do not interpret input properly for values of **ibase** that are greater than 16. This  
79384 is because numeric constants are recognized syntactically, rather than lexically, as described in  
79385 this volume of POSIX.1-2008. They are built from lexical tokens of single hexadecimal digits and  
79386 <period> characters. Since <blank> characters between tokens are not visible at the syntactic  
79387 level, it is not possible to recognize the multi-digit "digits" used in the higher bases properly.  
79388 The ability to recognize input in these bases was not considered useful enough to require  
79389 modifying these implementations. Note that the recognition of numeric constants at the  
79390 syntactic level is not a problem with conformance to this volume of POSIX.1-2008, as it does not  
79391 impact the behavior of conforming applications (and correct *bc* programs). Historical  
79392 implementations also accept input with all of the digits '*0*'–'*9*' and '*A*'–'*F*' regardless of the  
79393 value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate,  
79394 the behavior when they appear is undefined, except for the common case of:
- ```
79395 ibase=8;
79396     /* Process in octal base. */
79397 ...
79398 ibase=A
79399     /* Restore decimal base. */
```
- 79400 In some historical implementations, if the expression to be written is an uninitialized array

79401 element, a leading <space> and/or up to four leading 0 characters may be output before the  
79402 character zero. This behavior is considered a bug; it is unlikely that any currently conforming  
79403 application relies on:

79404 `echo 'b[3]' | bc`

79405 returning 00000 rather than 0.

79406 Exact calculation of the number of fractional digits to output for a given value in a base other  
79407 than 10 can be computationally expensive. Historical implementations use a faster  
79408 approximation, and this is permitted. Note that the requirements apply only to values of **obase**  
79409 that this volume of POSIX.1-2008 requires implementations to support (in particular, not to 1, 0,  
79410 or negative bases, if an implementation supports them as an extension).

79411 Historical implementations of *bc* did not allow array parameters to be passed as the last  
79412 parameter to a function. New implementations are encouraged to remove this restriction even  
79413 though it is not required by the grammar.

#### 79414 **FUTURE DIRECTIONS**

79415 None.

#### 79416 **SEE ALSO**

79417 [Section 1.3](#) (on page 2287), *awk*

79418 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 79419 **CHANGE HISTORY**

79420 First released in Issue 4.

#### 79421 **Issue 5**

79422 The FUTURE DIRECTIONS section is added.

#### 79423 **Issue 6**

79424 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several  
79425 interpretations of the ISO POSIX-2:1993 standard.

79426 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 79427 **Issue 7**

79428 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79429 **NAME**79430 `bg` — run jobs in the background79431 **SYNOPSIS**79432 UP `bg [job_id...]`79433 **DESCRIPTION**

79434 If job control is enabled (see the description of `set -m`), the `bg` utility shall resume suspended jobs  
 79435 from the current environment (see [Section 2.12](#), on page 2331) by running them as background  
 79436 jobs. If the job specified by `job_id` is already a running background job, the `bg` utility shall have  
 79437 no effect and shall exit successfully.

79438 Using `bg` to place a job into the background shall cause its process ID to become “known in the  
 79439 current shell execution environment”, as if it had been started as an asynchronous list; see  
 79440 [Section 2.9.3.1](#) (on page 2319).

79441 **OPTIONS**

79442 None.

79443 **OPERANDS**

79444 The following operand shall be supported:

79445 `job_id` Specify the job to be resumed as a background job. If no `job_id` operand is given,  
 79446 the most recently suspended job shall be used. The format of `job_id` is described in  
 79447 XBD [Section 3.203](#) (on page 65).

79448 **STDIN**

79449 Not used.

79450 **INPUT FILES**

79451 None.

79452 **ENVIRONMENT VARIABLES**79453 The following environment variables shall affect the execution of `bg`:

79454 `LANG` Provide a default value for the internationalization variables that are unset or null.  
 79455 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 79456 variables used to determine the values of locale categories.)

79457 `LC_ALL` If set to a non-empty string value, override the values of all the other  
 79458 internationalization variables.

79459 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
 79460 characters (for example, single-byte as opposed to multi-byte characters in  
 79461 arguments).

79462 `LC_MESSAGES`

79463 Determine the locale that should be used to affect the format and contents of  
 79464 diagnostic messages written to standard error.

79465 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

79466 **ASYNCHRONOUS EVENTS**

79467 Default.

79468 **STDOUT**79469 The output of `bg` shall consist of a line in the format:

79470 "[%d] %s\n", &lt;job-number&gt;, &lt;command&gt;

79471 where the fields are as follows:

79472 *<job-number>* A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using  
79473 these utilities, the job can be identified by prefixing the job number with ' % ' .

79474 *<command>* The associated command that was given to the shell.

#### 79475 **STDERR**

79476 The standard error shall be used only for diagnostic messages.

#### 79477 **OUTPUT FILES**

79478 None.

#### 79479 **EXTENDED DESCRIPTION**

79480 None.

#### 79481 **EXIT STATUS**

79482 The following exit values shall be returned:

79483 0 Successful completion.

79484 >0 An error occurred.

#### 79485 **CONSEQUENCES OF ERRORS**

79486 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the  
79487 background.

#### 79488 **APPLICATION USAGE**

79489 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see  
79490 XBD Chapter 11 (on page 199). At that point, *bg* can put the job into the background. This is  
79491 most effective when the job is expecting no terminal input and its output has been redirected to  
79492 non-terminal files. A background job can be forced to stop when it has terminal output by  
79493 issuing the command:

79494 `stty tostop`

79495 A background job can be stopped with the command:

79496 `kill -s stop job ID`

79497 The *bg* utility does not work as expected when it is operating in its own utility execution  
79498 environment because that environment has no suspended jobs. In the following examples:

79499 `... | xargs bg`  
79500 `(bg)`

79501 each *bg* operates in a different environment and does not share its parent shell's understanding  
79502 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

#### 79503 **EXAMPLES**

79504 None.

#### 79505 **RATIONALE**

79506 The extensions to the shell specified in this volume of POSIX.1-2008 have mostly been based on  
79507 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also  
79508 based on the KornShell. The standard developers examined the characteristics of the C shell  
79509 versions of these utilities and found that differences exist. Despite widespread use of the C shell,  
79510 the KornShell versions were selected for this volume of POSIX.1-2008 to maintain a degree of  
79511 uniformity with the rest of the KornShell features selected (such as the very popular command  
79512 line editing features).

- 79513 The *bg* utility is expected to wrap its output if the output exceeds the number of display  
79514 columns.
- 79515 **FUTURE DIRECTIONS**
- 79516 None.
- 79517 **SEE ALSO**
- 79518 [Section 2.9.3.1](#) (on page 2319), *fg*, *kill*, *jobs*, *wait*
- 79519 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199)
- 79520 **CHANGE HISTORY**
- 79521 First released in Issue 4.
- 79522 **Issue 6**
- 79523 This utility is marked as part of the User Portability Utilities option.
- 79524 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory  
79525 in this version. This is a FIPS requirement.
- 79526 **Issue 7**
- 79527 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 79528 NAME

79529 c99 — compile standard C programs

## 79530 SYNOPSIS

```
79531 CD c99 [options...] pathname [[pathname] [-I directory]
79532 [-L directory] [-l library]]...
```

## 79533 DESCRIPTION

79534 The *c99* utility is an interface to the standard C compilation system; it shall accept source code  
 79535 conforming to the ISO C standard. The system conceptually consists of a compiler and link  
 79536 editor. The input files referenced by *pathname* operands and *-I* option-arguments shall be  
 79537 compiled and linked to produce an executable file. (It is unspecified whether the linking occurs  
 79538 entirely within the operation of *c99*; some implementations may produce objects that are not  
 79539 fully resolved until the file is executed.)

79540 If the *-c* option is specified, for all *pathname* operands of the form *file.c*, the files:

79541  $\$(\text{basename } \textit{pathname} \textit{.c}) \textit{.o}$

79542 shall be created as the result of successful compilation. If the *-o* option is not specified, it is  
 79543 unspecified whether such *.o* files are created or deleted for the *file.c* operands.

79544 If there are no options that prevent link editing (such as *-c* or *-E*), and all input files compile and  
 79545 link without error, the resulting executable file shall be written according to the *-o outfile* option  
 79546 (if present) or to the file **a.out**.

79547 The executable file shall be created as specified in Section 1.1.1.4 (on page 2280), except that the  
 79548 file permission bits shall be set to:

79549 S\_IRWXO | S\_IRWXG | S\_IRWXU

79550 and the bits specified by the *umask* of the process shall be cleared.

## 79551 OPTIONS

79552 The *c99* utility shall conform to XBD Section 12.2 (on page 215), except that:

- 79553 • Options can be interspersed with operands.
- 79554 • The order of specifying the *-I*, *-L*, and *-l* options, and the order of specifying *-l* options  
 79555 with respect to *pathname* operands is significant.
- 79556 • Conforming applications shall specify each option separately; that is, grouping option  
 79557 letters (for example, *-cO*) need not be recognized by all implementations.

79558 The following options shall be supported:

79559 *-c* Suppress the link-edit phase of the compilation, and do not remove any object files  
 79560 that are produced.

79561 *-D name[=value]*

79562 Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of  
 79563 1 shall be used. The *-D* option has lower precedence than the *-U* option. That is, if  
 79564 *name* is used in both a *-U* and a *-D* option, *name* shall be undefined regardless of  
 79565 the order of the options. Additional implementation-defined *names* may be  
 79566 provided by the compiler. Implementations shall support at least 2048 bytes of *-D*  
 79567 definitions and 256 *names*.

- 79568        **-E**        Copy C-language source files to standard output, expanding all preprocessor  
79569        directives; no compilation shall be performed. If any operand is not a text file, the  
79570        effects are unspecified.
- 79571        **-g**        Produce symbolic information in the object or executable files; the nature of this  
79572        information is unspecified, and may be modified by implementation-defined  
79573        interactions with other options.
- 79574        **-I *directory***    Change the algorithm for searching for headers whose names are not absolute  
79575        pathnames to look in the *directory* named by the *directory* pathname before looking  
79576        in the usual places. Thus, headers whose names are enclosed in double-quotes (" ")  
79577        shall be searched for first in the directory of the file with the **#include** line, then in  
79578        directories named in **-I** options, and last in the usual places. For headers whose  
79579        names are enclosed in angle brackets ("**<**" "**>**"), the header shall be searched for only  
79580        in directories named in **-I** options and then in the usual places. Directories named  
79581        in **-I** options shall be searched in the order specified. Implementations shall  
79582        support at least ten instances of this option in a single **c99** command invocation.
- 79583        **-L *directory***    Change the algorithm of searching for the libraries named in the **-l** objects to look  
79584        in the *directory* named by the *directory* pathname before looking in the usual  
79585        places. Directories named in **-L** options shall be searched in the order specified.  
79586        Implementations shall support at least ten instances of this option in a single **c99**  
79587        command invocation. If a *directory* specified by a **-L** option contains files with  
79588        names starting with any of the strings "libc.", "libl.", "libpthread.",  
79589        "libm.", "librt.", "libtrace.", "libxnet.", or "liby.", the results are  
79590        unspecified.
- 79591        **-l *library***        Search the library named **lib*library*.a**. A library shall be searched when its name is  
79592        encountered, so the placement of a **-l** option is significant. Several standard  
79593        libraries can be specified in this manner, as described in the EXTENDED  
79594        DESCRIPTION section. Implementations may recognize implementation-defined  
79595        suffixes other than **.a** as denoting libraries.
- 79596        **-O *optlevel***        Specify the level of code optimization. If the *optlevel* option-argument is the digit  
79597        '0', all special code optimizations shall be disabled. If it is the digit '1', the  
79598        nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
79599        the system's default optimization is unspecified. It is unspecified whether code  
79600        generated in the presence of the **-O 0** option is the same as that generated when  
79601        **-O** is omitted. Other *optlevel* values may be supported.
- 79602        **-o *outfile***        Use the pathname *outfile*, instead of the default **a.out**, for the executable file  
79603        produced. If the **-o** option is present with **-c** or **-E**, the result is unspecified.
- 79604        **-s**        Produce object or executable files, or both, from which symbolic and other  
79605        information not required for proper execution using the *exec* family defined in the  
79606        System Interfaces volume of POSIX.1-2008 has been removed (stripped). If both **-g**  
79607        and **-s** options are present, the action taken is unspecified.
- 79608        **-U *name***        Remove any initial definition of *name*.
- 79609        Multiple instances of the **-D**, **-I**, **-L**, **-l**, and **-U** options can be specified.
- 79610        **OPERANDS**
- 79611        The application shall ensure that at least one *pathname* operand is specified. The following forms  
79612        for *pathname* operands shall be supported:

- 79613 *file.c* A C-language source file to be compiled and optionally linked. The application  
79614 shall ensure that the operand is of this form if the `-c` option is used.
- 79615 *file.a* A library of object files typically produced by the *ar* utility, and passed directly to  
79616 the link editor. Implementations may recognize implementation-defined suffixes  
79617 other than *.a* as denoting object file libraries.
- 79618 *file.o* An object file produced by *c99 -c* and passed directly to the link editor.  
79619 Implementations may recognize implementation-defined suffixes other than *.o* as  
79620 denoting object files.
- 79621 The processing of other files is implementation-defined.
- 79622 **STDIN**
- 79623 Not used.
- 79624 **INPUT FILES**
- 79625 The input file shall be one of the following: a text file containing a C-language source program,  
79626 an object file in the format produced by *c99 -c*, or a library of object files, in the format produced  
79627 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities  
79628 that produce files in these formats. Additional input file formats are implementation-defined.
- 79629 **ENVIRONMENT VARIABLES**
- 79630 The following environment variables shall affect the execution of *c99*:
- 79631 *LANG* Provide a default value for the internationalization variables that are unset or null.  
79632 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
79633 variables used to determine the values of locale categories.)
- 79634 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
79635 internationalization variables.
- 79636 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
79637 characters (for example, single-byte as opposed to multi-byte characters in  
79638 arguments and input files).
- 79639 *LC\_MESSAGES*
- 79640 Determine the locale that should be used to affect the format and contents of  
79641 diagnostic messages written to standard error.
- 79642 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 79643 *TMPDIR* Provide a pathname that should override the default directory for temporary files,  
79644 XSI if any. On XSI-conforming systems, provide a pathname that shall override the  
79645 default directory for temporary files, if any.
- 79646 **ASYNCHRONOUS EVENTS**
- 79647 Default.
- 79648 **STDOUT**
- 79649 If more than one *pathname* operand ending in *.c* (or possibly other unspecified suffixes) is given,  
79650 for each such file:
- 79651 `"%s:\n", <pathname>`
- 79652 may be written. These messages, if written, shall precede the processing of each input file; they  
79653 shall not be written to the standard output if they are written to the standard error, as described  
79654 in the `STDERR` section.
- 79655 If the `-E` option is specified, the standard output shall be a text file that represents the results of

79656 the preprocessing stage of the language; it may contain extra information appropriate for  
79657 subsequent compilation passes.

#### 79658 **STDERR**

79659 The standard error shall be used only for diagnostic messages. If more than one *pathname*  
79660 operand ending in *.c* (or possibly other unspecified suffixes) is given, for each such file:

79661 "%s:\n", <pathname>

79662 may be written to allow identification of the diagnostic and warning messages with the  
79663 appropriate input file. These messages, if written, shall precede the processing of each input file;  
79664 they shall not be written to the standard error if they are written to the standard output, as  
79665 described in the STDOUT section.

79666 This utility may produce warning messages about certain conditions that do not warrant  
79667 returning an error (non-zero) exit value.

#### 79668 **OUTPUT FILES**

79669 Object files or executable files or both are produced in unspecified formats. If the pathname of  
79670 an object file or executable file to be created by *c99* resolves to an existing directory entry for a  
79671 file that is not a regular file, it is unspecified whether *c99* shall attempt to create the file or shall  
79672 issue a diagnostic and exit with a non-zero exit status.

#### 79673 **EXTENDED DESCRIPTION**

##### 79674 **Standard Libraries**

79675 The *c99* utility shall recognize the following **-l** options for standard libraries:

79676 **-l c** This option shall make available all interfaces referenced in the System Interfaces  
79677 volume of POSIX.1-2008, with the possible exception of those interfaces listed as  
79678 residing in <aio.h>, <arpa/inet.h>, <complex.h>, <fenv.h>, <math.h>,  
79679 <mqueue.h>, <netdb.h>, <net/if.h>, <netinet/in.h>, <pthread.h>, <sched.h>,  
79680 <semaphore.h>, <spawn.h>, <sys/socket.h>, *pthread\_kill()*, and *pthread\_sigmask()*  
79681 in <signal.h>, <trace.h>, interfaces marked as optional in <sys/mman.h>,  
79682 interfaces marked as ADV (Advisory Information) in <fcntl.h>, and interfaces  
79683 beginning with the prefix *clock\_* or *time\_* in <time.h>. This option shall not be  
79684 required to be present to cause a search of this library.

79685 **-l l** This option shall make available all interfaces required by the C-language output  
79686 of *lex* that are not made available through the **-l c** option.

79687 **-l pthread** This option shall make available all interfaces referenced in <pthread.h> and  
79688 *pthread\_kill()* and *pthread\_sigmask()* referenced in <signal.h>. An implementation  
79689 may search this library in the absence of this option.

79690 **-l m** This option shall make available all interfaces referenced in <math.h>,  
79691 <complex.h>, and <fenv.h>. An implementation may search this library in the  
79692 absence of this option.

79693 **-l rt** This option shall make available all interfaces referenced in <aio.h>, <mqueue.h>,  
79694 <sched.h>, <semaphore.h>, and <spawn.h>, interfaces marked as optional in  
79695 <sys/mman.h>, interfaces marked as ADV (Advisory Information) in <fcntl.h>,  
79696 and interfaces beginning with the prefix *clock\_* and *time\_* in <time.h>. An  
79697 implementation may search this library in the absence of this option.

- 79698 OB **-l trace** This option shall make available all interfaces referenced in `<trace.h>`. An implementation may search this library in the absence of this option.
- 79699
- 79700 **-l xnet** This option shall make available all interfaces referenced in `<arpa/inet.h>`, `<netdb.h>`, `<net/if.h>`, `<netinet/in.h>`, and `<sys/socket.h>`. An implementation may search this library in the absence of this option.
- 79701
- 79702
- 79703 **-l y** This option shall make available all interfaces required by the C-language output of *yacc* that are not made available through the `-l c` option.
- 79704

79705 In the absence of options that inhibit invocation of the link editor, such as `-c` or `-E`, the *c99* utility shall cause the equivalent of a `-l c` option to be passed to the link editor after the last *pathname* operand or `-l` option, causing it to be searched after all other object files and libraries are loaded.

79706

79707

79708 OB It is unspecified whether the libraries `libc.a`, `libl.a`, `libm.a`, `libpthread.a`, `librt.a`, `libtrace.a`, `libxnet.a`, or `liby.a` exist as regular files. The implementation may accept as `-l` option-arguments names of objects that do not exist as regular files.

79709

79710

79711 **External Symbols**

79712 The C compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

79713

79714

79715 The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be written to the standard output if the implementation-defined limit is exceeded; other actions are unspecified.

79716

79717

79718

79719 **Programming Environments**

79720 All implementations shall support one of the following programming environments as a default. Implementations may support more than one of the following programming environments. Applications can use *sysconf()* or *getconf* to determine which programming environments are supported.

79721

79722

79723

79724 **Table 4-4** Programming Environments: Type Sizes

| Programming Environment<br><i>getconf</i> Name | Bits in<br>int | Bits in<br>long | Bits in<br>pointer | Bits in<br>off_t |
|------------------------------------------------|----------------|-----------------|--------------------|------------------|
| _POSIX_V7_ILP32_OFF32                          | 32             | 32              | 32                 | 32               |
| _POSIX_V7_ILP32_OFFBIG                         | 32             | 32              | 32                 | ≥64              |
| _POSIX_V7_LP64_OFF64                           | 32             | 64              | 64                 | 64               |
| _POSIX_V7_LP64_OFFBIG                          | ≥32            | ≥64             | ≥64                | ≥64              |

79731 All implementations shall support one or more environments where the widths of the following types are no greater than the width of type `long`:

79732

- 79733 `blksize_t` `ptrdiff_t` `tcflag_t`
- 79734 `cc_t` `size_t` `wchar_t`
- 79735 `mode_t` `speed_t` `wint_t`
- 79736 `nfds_t` `ssize_t`
- 79737 `pid_t` `suseconds_t`

79738 The executable files created when these environments are selected shall be in a proper format for

79739 execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4  
 79740 (on page 2492), or it may be another environment. The names for the environments that meet  
 79741 this requirement shall be output by a *getconf* command using the  
 79742 POSIX\_V7\_WIDTH\_RESTRICTED\_ENVS argument, as a <newline>-separated list of names  
 79743 suitable for use with the *getconf* -v option. If more than one environment meets the requirement,  
 79744 the names of all such environments shall be output on separate lines. Any of these names can  
 79745 then be used in a subsequent *getconf* command to obtain the flags specific to that environment  
 79746 with the following suffixes added as appropriate:

79747 `_CFLAGS` To get the C compiler flags.

79748 `_LDFLAGS` To get the linker/loader flags.

79749 `_LIBS` To get the libraries.

79750 This requirement may be removed in a future version.

79751 When this utility processes a file containing a function called *main()*, it shall be defined with a  
 79752 return type equivalent to `int`. Using return from the initial call to *main()* shall be equivalent  
 79753 (other than with respect to language scope issues) to calling *exit()* with the returned value.  
 79754 Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The  
 79755 implementation shall not declare a prototype for this function.

79756 Implementations provide configuration strings for C compiler flags, linker/loader flags, and  
 79757 libraries for each supported environment. When an application needs to use a specific  
 79758 programming environment rather than the implementation default programming environment  
 79759 while compiling, the application shall first verify that the implementation supports the desired  
 79760 environment. If the desired programming environment is supported, the application shall then  
 79761 invoke *c99* with the appropriate C compiler flags as the first options for the compile, the  
 79762 appropriate linker/loader flags after any other options except `-I` but before any operands or `-I`  
 79763 options, and the appropriate libraries at the end of the operands and `-I` options.

79764 Conforming applications shall not attempt to link together object files compiled for different  
 79765 programming models. Applications shall also be aware that binary data placed in shared  
 79766 memory or in files might not be recognized by applications built for other programming models.

79767 **Table 4-5** Programming Environments: *c99* Arguments

| 79768<br>79769          | Programming Environment<br><i>getconf</i> Name | Use                                                  | <i>c99</i> Arguments<br><i>getconf</i> Name                                                                                        |
|-------------------------|------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 79770<br>79771<br>79772 | <code>_POSIX_V7_ILP32_OFF32</code>             | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V7_ILP32_OFF32_CFLAGS</code><br><code>POSIX_V7_ILP32_OFF32_LDFLAGS</code><br><code>POSIX_V7_ILP32_OFF32_LIBS</code>    |
| 79773<br>79774<br>79775 | <code>_POSIX_V7_ILP32_OFFBIG</code>            | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V7_ILP32_OFFBIG_CFLAGS</code><br><code>POSIX_V7_ILP32_OFFBIG_LDFLAGS</code><br><code>POSIX_V7_ILP32_OFFBIG_LIBS</code> |
| 79776<br>79777<br>79778 | <code>_POSIX_V7_LP64_OFF64</code>              | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V7_LP64_OFF64_CFLAGS</code><br><code>POSIX_V7_LP64_OFF64_LDFLAGS</code><br><code>POSIX_V7_LP64_OFF64_LIBS</code>       |
| 79779<br>79780<br>79781 | <code>_POSIX_V7_LP64_OFFBIG</code>             | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V7_LP64_OFFBIG_CFLAGS</code><br><code>POSIX_V7_LP64_OFFBIG_LDFLAGS</code><br><code>POSIX_V7_LP64_OFFBIG_LIBS</code>    |

79782 In addition to the type size programming environments above, all implementations also support

79783 a multi-threaded programming environment that is orthogonal to all of the programming  
79784 environments listed above. The *getconf* utility can be used to get flags for the threaded  
79785 programming environment, as indicated in Table 4-6.

79786 **Table 4-6** Threaded Programming Environment: *c99* Arguments

| 79787<br>79788 | Programming Environment<br><i>getconf</i> Name | Use                                     | <i>c99</i> Arguments<br><i>getconf</i> Name         |
|----------------|------------------------------------------------|-----------------------------------------|-----------------------------------------------------|
| 79789<br>79790 | _POSIX_THREADS                                 | C Compiler Flags<br>Linker/Loader Flags | POSIX_V7_THREADS_CFLAGS<br>POSIX_V7_THREADS_IDFLAGS |

79791 These programming environment flags may be used in conjunction with any of the type size  
79792 programming environments supported by the implementation.

#### 79793 EXIT STATUS

79794 The following exit values shall be returned:

- 79795 0 Successful compilation or link edit.
- 79796 >0 An error occurred.

#### 79797 CONSEQUENCES OF ERRORS

79798 When *c99* encounters a compilation error that causes an object file not to be created, it shall write  
79799 a diagnostic to standard error and continue to compile other source code operands, but it shall  
79800 not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a  
79801 diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A  
79802 conforming application shall rely on the exit status of *c99*, rather than on the existence or mode  
79803 of the executable file.

#### 79804 APPLICATION USAGE

79805 Since the *c99* utility usually creates files in the current directory during the compilation process,  
79806 it is typically necessary to run the *c99* utility in a directory in which a file can be created.

79807 On systems providing POSIX Conformance (see XBD Chapter 2, on page 15), *c99* is required  
79808 only with the C-Language Development option; XSI-conformant systems always provide *c99*.

79809 Some historical implementations have created *.o* files when *-c* is not specified and more than  
79810 one source file is given. Since this area is left unspecified, the application cannot rely on *.o* files  
79811 being created, but it also must be prepared for any related *.o* files that already exist being deleted  
79812 at the completion of the link edit.

79813 There is the possible implication that if a user supplies versions of the standard functions (before  
79814 they would be encountered by an implicit *-l c* or explicit *-l m*), that those versions would be  
79815 used in place of the standard versions. There are various reasons this might not be true  
79816 (functions defined as macros, manipulations for clean name space, and so on), so the existence of  
79817 files named in the same manner as the standard libraries within the *-L* directories is explicitly  
79818 stated to produce unspecified behavior.

79819 All of the functions specified in the System Interfaces volume of POSIX.1-2008 may be made  
79820 visible by implementations when the Standard C Library is searched. Conforming applications  
79821 must explicitly request searching the other standard libraries when functions made visible by  
79822 those libraries are used.

79823 In the ISO C standard the mapping from physical source characters to the C source character set  
79824 is implementation-defined. Implementations may strip white-space characters before the  
79825 terminating <newline> of a (physical) line as part of this mapping and, as a consequence of this,

79826 one or more white-space characters (and no other characters) between a <backslash> character  
 79827 and the <newline> character that terminates the line produces implementation-defined results.  
 79828 Portable applications should not use such constructs.

79829 Some *c99* compilers not conforming to POSIX.1-2008 do not support trigraphs by default.

### 79830 EXAMPLES

79831 1. The following usage example compiles **foo.c** and creates the executable file **foo**:

```
79832 c99 -o foo foo.c
```

79833 The following usage example compiles **foo.c** and creates the object file **foo.o**:

```
79834 c99 -c foo.c
```

79835 The following usage example compiles **foo.c** and creates the executable file **a.out**:

```
79836 c99 foo.c
```

79837 The following usage example compiles **foo.c**, links it with **bar.o**, and creates the  
 79838 executable file **a.out**. It may also create and leave **foo.o**:

```
79839 c99 foo.c bar.o
```

79840 2. The following example shows how an application using threads interfaces can test for  
 79841 support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer**  
 79842 types and an **off\_t** type using at least 64 bits:

```
79843 offbig_env=$(getconf _POSIX_V7_ILP32_OFFBIG)
79844 if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
79845 then
79846     c99 $(getconf POSIX_V7_ILP32_OFFBIG_CFLAGS) \
79847         $(getconf POSIX_V7_THREADS_CFLAGS) -D_XOPEN_SOURCE=700 \
79848         $(getconf POSIX_V7_ILP32_OFFBIG_LDFLAGS) \
79849         $(getconf POSIX_V7_THREADS_LDFLAGS) foo.c -o foo \
79850         $(getconf POSIX_V7_ILP32_OFFBIG_LIBS) \
79851         -l pthread
79852 else
79853     echo ILP32_OFFBIG programming environment not supported
79854     exit 1
79855 fi
```

79856 3. The following examples clarify the use and interactions of **-L** and **-l** options.

79857 Consider the case in which module **a.c** calls function *f()* in library **libQ.a**, and module **b.c**  
 79858 calls function *g()* in library **libp.a**. Assume that both libraries reside in **/a/b/c**. The  
 79859 command line to compile and link in the desired way is:

```
79860 c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

79861 In this case the **-L** option need only precede the first **-l** option, since both **libQ.a** and  
 79862 **libp.a** reside in the same directory.

79863 Multiple **-L** options can be used when library name collisions occur. Building on the  
 79864 previous example, suppose that the user wants to use a new **libp.a**, in **/a/a/a**, but still  
 79865 wants *f()* from **/a/b/c/libQ.a**:

```
79866 c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

79867 In this example, the linker searches the **-L** options in the order specified, and finds

79868 */a/a/a/libp.a* before */a/b/c/libp.a* when resolving references for **b.c**. The order of the **-l**  
 79869 options is still important, however.

79870 4. The following example shows how an application can use a programming environment  
 79871 where the widths of the following types:

79872 **blksize\_t, cc\_t, mode\_t, nfd\_t, pid\_t, ptrdiff\_t, size\_t, speed\_t, ssize\_t, suseconds\_t,**  
 79873 **tcflag\_t, wchar\_t, wint\_t**

79874 are no greater than the width of type **long**:

79875 # First choose one of the listed environments ...

79876 # ... if there are no additional constraints, the first one will do:  
 79877 CENV=\$(getconf POSIX\_V7\_WIDTH\_RESTRICTED\_ENVS | head -n 1)

79878 # ... or, if an environment that supports large files is preferred,  
 79879 # look for names that contain "OFF64" or "OFFBIG". (This chooses  
 79880 # the last one in the list if none match.)  
 79881 for CENV in \$(getconf POSIX\_V7\_WIDTH\_RESTRICTED\_ENVS)  
 79882 do  
 79883 case \$CENV in  
 79884 \*OFF64\*|\*OFFBIG\*) break ;;  
 79885 esac  
 79886 done

79887 # The chosen environment name can now be used like this:

79888 c99 \$(getconf \${CENV}\_CFLAGS) -D \_POSIX\_C\_SOURCE=200809L \  
 79889 \$(getconf \${CENV}\_LD\_FLAGS) foo.c -o foo \  
 79890 \$(getconf \${CENV}\_LIBS)

#### 79891 RATIONALE

79892 The **c99** utility is based on the **c89** utility originally introduced in the ISO POSIX-2:1993  
 79893 standard.

79894 Some of the changes from **c89** include the ability to intersperse options and operands (which  
 79895 many **c89** implementations allowed despite it not being specified), the description of **-l** as an  
 79896 option instead of an operand, and the modification to the contents of the Standard Libraries  
 79897 section to account for new headers and options; for example, **<spawn.h>** added to the  
 79898 description of **-l rt**, and **-l trace** added for the Tracing option.

79899 POSIX.1-2008 specifies that the **c99** utility must be able to use regular files for **\*.o** files and for  
 79900 **a.out** files. Implementations are free to overwrite existing files of other types when attempting to  
 79901 create object files and executable files, but are not required to do so. If something other than a  
 79902 regular file is specified and using it fails for any reason, **c99** is required to issue a diagnostic  
 79903 message and exit with a non-zero exit status. But for some file types, the problem may not be  
 79904 noticed for a long time. For example, if a FIFO named **a.out** exists in the current directory, **c99**  
 79905 may attempt to open **a.out** and will hang in the *open()* call until another process opens the FIFO  
 79906 for reading. Then **c99** may write most of the **a.out** to the FIFO and fail when it tries to seek back  
 79907 close to the start of the file to insert a timestamp (FIFOs are not seekable files). The **c99** utility is  
 79908 also allowed to issue a diagnostic immediately if it encounters an **a.out** or **\*.o** file that is not a  
 79909 regular file. For portable use, applications should ensure that any **a.out**, **-o** option-argument, or  
 79910 **\*.o** files corresponding to any **\*.c** files do not conflict with names already in use that are not  
 79911 regular files or symbolic links that point to regular files.

79912 On many systems, multi-threaded applications run in a programming environment that is

79913 distinct from that used by single-threaded applications. This multi-threaded programming  
 79914 environment (in addition to needing to specify `-l pthread` at link time) may require additional  
 79915 flags to be set when headers are processed at compile time (`-D_REENTRANT` being common).  
 79916 This programming environment is orthogonal to the type size programming environments  
 79917 discussed above and listed in [Table 4-4](#) (on page 2492). This version of the standard adds `getconf`  
 79918 utility calls to provide the C compiler flags and linker/loader flags needed to support multi-  
 79919 threaded applications. Note that on a system where single-threaded applications are a special  
 79920 case of a multi-threaded application, both of these `getconf` calls may return NULL strings, on  
 79921 other implementations both of these strings may be non-NULL strings.

79922 The C standardization committee invented trigraphs (e.g., "??!" to represent '~() to address  
 79923 character portability problems in development environments based on national variants of the  
 79924 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already  
 79925 obsolete by the time the first ISO C standard was published, and in practice trigraphs have not  
 79926 been used for their intended purpose, and usually are intended to have their original meaning in  
 79927 K&R C. For example, in practice a C-language source string like "What??!" is usually intended  
 79928 to end in two <question-mark> characters and an <exclamation-mark>, not in '~() .

#### 79929 FUTURE DIRECTIONS

79930 None.

#### 79931 SEE ALSO

79932 [Section 1.1.1.4](#) (on page 2280), [ar](#), [getconf](#), [make](#), [nm](#), [strip](#), [umask](#)

79933 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215), [Chapter 13](#) (on page 219)

79934 XSH [exec](#), [sysconf\(\)](#)

#### 79935 CHANGE HISTORY

79936 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

79937 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/12 is applied, correcting the EXTENDED  
 79938 DESCRIPTION of `-l c` and `-l m`. Previously, the text did not take into account the presence of  
 79939 the `c99` math headers.

79940 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/13 is applied, changing the reference to  
 79941 the `libxnet` library to `libxnet.a`.

79942 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/5 is applied, updating the OPTIONS  
 79943 section, so that the names of files contained in the directory specified by the `-L` option are not  
 79944 assumed to end in the `.a` suffix. The set of library prefixes is also updated.

79945 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/6 is applied, removing the lead  
 79946 underscore from the `POSIX_V6_WIDTH_RESTRICTED_ENVS` variable in the EXTENDED  
 79947 DESCRIPTION and the EXAMPLES sections.

#### 79948 Issue 7

79949 Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding to the  
 79950 OUTPUT FILES section and also adding associated RATIONALE.

79951 Austin Group Interpretation 1003.1-2001 #095 is applied, clarifying the `-l library` operand.

79952 Austin Group Interpretation 1003.1-2001 #166 is applied.

79953 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the handling of trailing  
 79954 white-space characters.

79955 Austin Group Interpretation 1003.1-2001 #191 is applied, adding APPLICATION USAGE and  
 79956 RATIONALE regarding C-language trigraphs.

- 79957 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
79958 apply (options can be interspersed with operands).
- 79959 SD5-XCU-ERN-11 is applied, adding the `<net/if.h>` header to the descriptions of `-l c` and  
79960 `-l xnet`.
- 79961 SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.
- 79962 SD5-XCU-ERN-67 and SD5-XCU-ERN-97 are applied, updating the SYNOPSIS.
- 79963 SD5-XCU-ERN-133 is applied, updating the EXTENDED DESCRIPTION.
- 79964 The *getconf* variables for the supported programming environments are updated to be V7.
- 79965 The `-l trace` operand is marked obsolescent.
- 79966 The *c99* reference page is rewritten to describe `-l` as an option rather than an operand.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

79967 **NAME**

79968 cal — print a calendar

79969 **SYNOPSIS**79970 XSI cal `[[month] year]`79971 **DESCRIPTION**

79972 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from  
 79973 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,  
 79974 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on  
 79975 September 14, 1752.

79976 **OPTIONS**

79977 None.

79978 **OPERANDS**

79979 The following operands shall be supported:

79980 *month* Specify the month to be displayed, represented as a decimal integer from 1  
 79981 (January) to 12 (December). The default shall be the current month.

79982 *year* Specify the year for which the calendar is displayed, represented as a decimal  
 79983 integer from 1 to 9999. The default shall be the current year.

79984 **STDIN**

79985 Not used.

79986 **INPUT FILES**

79987 None.

79988 **ENVIRONMENT VARIABLES**79989 The following environment variables shall affect the execution of *cal*:

79990 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 79991 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 79992 variables used to determine the values of locale categories.)

79993 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 79994 internationalization variables.

79995 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 79996 characters (for example, single-byte as opposed to multi-byte characters in  
 79997 arguments).

79998 *LC\_MESSAGES*

79999 Determine the locale that should be used to affect the format and contents of  
 80000 diagnostic messages written to standard error, and informative messages written  
 80001 to standard output.

80002 *LC\_TIME* Determine the format and contents of the calendar.

80003 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

80004 *TZ* Determine the timezone used to calculate the value of the current month.

80005 **ASYNCHRONOUS EVENTS**

80006 Default.

- 80007 **STDOUT**  
80008 The standard output shall be used to display the calendar, in an unspecified format.
- 80009 **STDERR**  
80010 The standard error shall be used only for diagnostic messages.
- 80011 **OUTPUT FILES**  
80012 None.
- 80013 **EXTENDED DESCRIPTION**  
80014 None.
- 80015 **EXIT STATUS**  
80016 The following exit values shall be returned:  
80017 0 Successful completion.  
80018 >0 An error occurred.
- 80019 **CONSEQUENCES OF ERRORS**  
80020 Default.
- 80021 **APPLICATION USAGE**  
80022 Note that:  
80023 cal 83  
80024 refers to A.D. 83, not 1983.
- 80025 **EXAMPLES**  
80026 None.
- 80027 **RATIONALE**  
80028 None.
- 80029 **FUTURE DIRECTIONS**  
80030 A future version of this standard may support locale-specific recognition of the date of adoption  
80031 of the Gregorian calendar.
- 80032 **SEE ALSO**  
80033 XBD [Chapter 8](#) (on page 173)
- 80034 **CHANGE HISTORY**  
80035 First released in Issue 2.
- 80036 **Issue 6**  
80037 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of  
80038 the Gregorian calendar.
- 80039 **Issue 7**  
80040 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80041 **NAME**

80042           cat — concatenate and print files

80043 **SYNOPSIS**80044           cat [-u] [*file...*]80045 **DESCRIPTION**80046           The *cat* utility shall read files in sequence and shall write their contents to the standard output in  
80047           the same sequence.80048 **OPTIONS**80049           The *cat* utility shall conform to XBD [Section 12.2](#) (on page 215).

80050           The following option shall be supported:

80051           **-u**           Write bytes from the input file to the standard output without delay as each is  
80052           read.80053 **OPERANDS**

80054           The following operand shall be supported:

80055           *file*           A pathname of an input file. If no *file* operands are specified, the standard input  
80056           shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at  
80057           that point in the sequence. The *cat* utility shall not close and reopen standard input  
80058           when it is referenced in this way, but shall accept multiple occurrences of '-' as a  
80059           *file* operand.80060 **STDIN**80061           The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
80062           See the INPUT FILES section.80063 **INPUT FILES**

80064           The input files can be any file type.

80065 **ENVIRONMENT VARIABLES**80066           The following environment variables shall affect the execution of *cat*:80067           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
80068           (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
80069           variables used to determine the values of locale categories.)80070           **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
80071           internationalization variables.80072           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
80073           characters (for example, single-byte as opposed to multi-byte characters in  
80074           arguments).80075           **LC\_MESSAGES**80076           Determine the locale that should be used to affect the format and contents of  
80077           diagnostic messages written to standard error.80078           **XSI**           **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.80079 **ASYNCHRONOUS EVENTS**

80080           Default.

80081 **STDOUT**

80082 The standard output shall contain the sequence of bytes read from the input files. Nothing else  
80083 shall be written to the standard output.

80084 **STDERR**

80085 The standard error shall be used only for diagnostic messages.

80086 **OUTPUT FILES**

80087 None.

80088 **EXTENDED DESCRIPTION**

80089 None.

80090 **EXIT STATUS**

80091 The following exit values shall be returned:

80092 0 All input files were output successfully.

80093 >0 An error occurred.

80094 **CONSEQUENCES OF ERRORS**

80095 Default.

80096 **APPLICATION USAGE**

80097 The **-u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support  
80098 the following sequence:

```
80099 mkfifo foo
80100 cat -u foo > /dev/tty13 &
80101 cat -u > foo
```

80102 It is unspecified whether standard output is or is not buffered in the default case. This is  
80103 sometimes of interest when standard output is associated with a terminal, since buffering may  
80104 delay the output. The presence of the **-u** option guarantees that unbuffered I/O is available. It is  
80105 implementation-defined whether the *cat* utility buffers output if the **-u** option is not specified.  
80106 Traditionally, the **-u** option is implemented using the equivalent of the *setvbuf()* function  
80107 defined in the System Interfaces volume of POSIX.1-2008.

80108 **EXAMPLES**

80109 The following command:

```
80110 cat myfile
```

80111 writes the contents of the file **myfile** to standard output.

80112 The following command:

```
80113 cat doc1 doc2 > doc.all
```

80114 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

80115 Because of the shell language mechanism used to perform output redirection, a command such  
80116 as this:

```
80117 cat doc doc.end > doc
```

80118 causes the original data in **doc** to be lost.

80119 The command:

```
80120 cat start - middle - end > file
```

80121 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a

80122 single invocation of *cat*. Note, however, that if standard input is a regular file, this would be  
80123 equivalent to the command:

```
80124 cat start - middle /dev/null end > file
```

80125 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a  
80126 file operand and an end-of-file condition would be detected immediately when '-' was  
80127 referenced the second time.

#### 80128 RATIONALE

80129 Historical versions of the *cat* utility include the *-e*, *-t*, and *-v*, options which permit the ends of  
80130 lines, <tab> characters, and invisible characters, respectively, to be rendered visible in the  
80131 output. The standard developers omitted these options because they provide too fine a degree of  
80132 control over what is made visible, and similar output can be obtained using a command such as:

```
80133 sed -n l pathname
```

80134 The latter also has the advantage that its output is unambiguous, whereas the output of  
80135 historical *cat -etv* is not.

80136 The *-s* option was omitted because it corresponds to different functions in BSD and System  
80137 V-based systems. The BSD *-s* option to squeeze blank lines can be accomplished by the shell  
80138 script shown in the following example:

```
80139 sed -n '  
80140 # Write non-empty lines.  
80141 ./ {  
80142     p  
80143     d  
80144 }  
80145 # Write a single empty line, then look for more empty lines.  
80146 /^$/ p  
80147 # Get next line, discard the held <newline> (empty line),  
80148 # and look for more empty lines.  
80149 :Empty  
80150 /^$/ {  
80151     N  
80152     s/./ /  
80153     b Empty  
80154 }  
80155 # Write the non-empty line before going back to search  
80156 # for the first in a set of empty lines.  
80157     p  
80158 '
```

80159 The System V *-s* option to silence error messages can be accomplished by redirecting the  
80160 standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the  
80161 same as the POSIX "empty line": a line consisting only of a <newline>.

80162 The BSD *-n* option was omitted because similar functionality can be obtained from the *-n*  
80163 option of the *pr* utility.

#### 80164 FUTURE DIRECTIONS

80165 None.

80166 **SEE ALSO**80167 [more](#)80168 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)80169 XSH [setobuf\(\)](#)80170 **CHANGE HISTORY**

80171 First released in Issue 2.

80172 **Issue 7**

80173 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80174 SD5-XCU-ERN-174 is applied, changing the RATIONALE.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

80175 **NAME**

80176 cd — change the working directory

80177 **SYNOPSIS**80178 cd [-L|-P] [*directory*]

80179 cd -

80180 **DESCRIPTION**

80181 The *cd* utility shall change the working directory of the current shell execution environment (see  
 80182 [Section 2.12](#), on page 2331) by executing the following steps in sequence. (In the following steps,  
 80183 the symbol **curpath** represents an intermediate value used to simplify the description of the  
 80184 algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

- 80185 1. If no *directory* operand is given and the *HOME* environment variable is empty or  
 80186 undefined, the default behavior is implementation-defined and no further steps shall be  
 80187 taken.
- 80188 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty  
 80189 value, the *cd* utility shall behave as if the directory named in the *HOME* environment  
 80190 variable was specified as the *directory* operand.
- 80191 3. If the *directory* operand begins with a <slash> character, set **curpath** to the operand and  
 80192 proceed to step 7.
- 80193 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 80194 5. Starting with the first pathname in the <colon>-separated pathnames of *CDPATH* (see the  
 80195 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the  
 80196 concatenation of that pathname, a <slash> character if that pathname did not end with a  
 80197 <slash> character, and the *directory* operand names a directory. If the pathname is null,  
 80198 test if the concatenation of dot, a <slash> character, and the operand names a directory. In  
 80199 either case, if the resulting string names an existing directory, set **curpath** to that string  
 80200 and proceed to step 7. Otherwise, repeat this step with the next pathname in *CDPATH*  
 80201 until all pathnames have been tested.
- 80202 6. If the **-P** option is in effect, set **curpath** to the *directory* operand. Otherwise, set **curpath** to  
 80203 the string formed by the concatenation of the value of *PWD*, a <slash> character if the  
 80204 value of *PWD* did not end with a <slash> character, and the operand.
- 80205 7. If the **-P** option is in effect, proceed to step 10. If **curpath** does not begin with a <slash>  
 80206 character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a  
 80207 <slash> character if the value of *PWD* did not end with a <slash> character, and **curpath**.
- 80208 8. The **curpath** value shall then be converted to canonical form as follows, considering each  
 80209 component from beginning to end, in sequence:
  - 80210 a. Dot components and any <slash> characters that separate them from the next  
 80211 component shall be deleted.
  - 80212 b. For each dot-dot component, if there is a preceding component and it is neither  
 80213 root nor dot-dot, then:
    - 80214 i. If the preceding component does not refer (in the context of pathname  
 80215 resolution with symbolic links followed) to a directory, then the *cd* utility  
 80216 shall display an appropriate error message and no further steps shall be  
 80217 taken.

- 80218 ii. The preceding component, all <slash> characters separating the preceding  
80219 component from dot-dot, dot-dot, and all <slash> characters separating dot-  
80220 dot from the following component (if any) shall be deleted.
- 80221 c. An implementation may further simplify **curpath** by removing any trailing  
80222 <slash> characters that are not also leading <slash> characters, replacing multiple  
80223 non-leading consecutive <slash> characters with a single <slash>, and replacing  
80224 three or more leading <slash> characters with a single <slash>. If, as a result of  
80225 this canonicalization, the **curpath** variable is null, no further steps shall be taken.
- 80226 9. If **curpath** is longer than {PATH\_MAX} bytes (including the terminating null) and the  
80227 *directory* operand was not longer than {PATH\_MAX} bytes (including the terminating  
80228 null), then **curpath** shall be converted from an absolute pathname to an equivalent  
80229 relative pathname if possible. This conversion shall always be considered possible if the  
80230 value of *PWD*, with a trailing <slash> added if it does not already have one, is an initial  
80231 substring of **curpath**. Whether or not it is considered possible under other circumstances  
80232 is unspecified. Implementations may also apply this conversion if **curpath** is not longer  
80233 than {PATH\_MAX} bytes or the *directory* operand was longer than {PATH\_MAX} bytes.
- 80234 10. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with  
80235 **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall  
80236 display an appropriate error message and the remainder of this step shall not be  
80237 executed. If the **-P** option is not in effect, the *PWD* environment variable shall be set to  
80238 the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative  
80239 pathname). If the **-P** option is in effect, the *PWD* environment variable shall be set to the  
80240 string that would be output by *pwd -P*. If there is insufficient permission on the new  
80241 directory, or on any parent of that directory, to determine the current working directory,  
80242 the value of the *PWD* environment variable is unspecified.

80243 If, during the execution of the above steps, the *PWD* environment variable is changed, the  
80244 *OLDPWD* environment variable shall also be changed to the value of the old working directory  
80245 (that is the current working directory immediately prior to the call to *cd*).

#### 80246 OPTIONS

80247 The *cd* utility shall conform to XBD Section 12.2 (on page 215).

80248 The following options shall be supported by the implementation:

- 80249 **-L** Handle the operand dot-dot logically; symbolic link components shall not be  
80250 resolved before dot-dot components are processed (see steps 8. and 9. in the  
80251 DESCRIPTION).
- 80252 **-P** Handle the operand dot-dot physically; symbolic link components shall be  
80253 resolved before dot-dot components are processed (see step 7. in the  
80254 DESCRIPTION).

80255 If both **-L** and **-P** options are specified, the last of these options shall be used and all others  
80256 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the  
80257 DESCRIPTION.

#### 80258 OPERANDS

80259 The following operands shall be supported:

- 80260 *directory* An absolute or relative pathname of the directory that shall become the new  
80261 working directory. The interpretation of a relative pathname by *cd* depends on the  
80262 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an  
80263 empty string, the results are unspecified.

- 80264 – When a <hyphen> is used as the operand, this shall be equivalent to the command:
- 80265 `cd "$OLDPWD" && pwd`
- 80266 which changes to the previous working directory and then writes its name.
- 80267 **STDIN**
- 80268 Not used.
- 80269 **INPUT FILES**
- 80270 None.
- 80271 **ENVIRONMENT VARIABLES**
- 80272 The following environment variables shall affect the execution of *cd*:
- 80273 *CDPATH* A <colon>-separated list of pathnames that refer to directories. The *cd* utility shall
- 80274 use this list in its attempt to change the directory, as described in the
- 80275 DESCRIPTION. An empty string in place of a directory pathname represents the
- 80276 current directory. If *CDPATH* is not set, it shall be treated as if it were an empty
- 80277 string.
- 80278 *HOME* The name of the directory, used when no *directory* operand is specified.
- 80279 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 80280 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
- 80281 variables used to determine the values of locale categories.)
- 80282 *LC\_ALL* If set to a non-empty string value, override the values of all the other
- 80283 internationalization variables.
- 80284 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 80285 characters (for example, single-byte as opposed to multi-byte characters in
- 80286 arguments).
- 80287 *LC\_MESSAGES*
- 80288 Determine the locale that should be used to affect the format and contents of
- 80289 diagnostic messages written to standard error.
- 80290 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 80291 *OLDPWD* A pathname of the previous working directory, used by *cd -*.
- 80292 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets
- 80293 or unsets the value of *PWD*, the behavior of *cd* is unspecified.
- 80294 **ASYNCHRONOUS EVENTS**
- 80295 Default.
- 80296 **STDOUT**
- 80297 If a non-empty directory name from *CDPATH* is used, or if *cd -* is used, an absolute pathname of
- 80298 the new working directory shall be written to the standard output as follows:
- 80299 `"%s\n", <new directory>`
- 80300 Otherwise, there shall be no output.
- 80301 **STDERR**
- 80302 The standard error shall be used only for diagnostic messages.

**cd**

Utilities

80303 **OUTPUT FILES**

80304 None.

80305 **EXTENDED DESCRIPTION**

80306 None.

80307 **EXIT STATUS**

80308 The following exit values shall be returned:

80309 0 The directory was successfully changed.

80310 &gt;0 An error occurred.

80311 **CONSEQUENCES OF ERRORS**

80312 The working directory shall remain unchanged.

80313 **APPLICATION USAGE**

80314 Since *cd* affects the current shell execution environment, it is always provided as a shell regular  
 80315 built-in. If it is called in a subshell or separate utility execution environment, such as one of the  
 80316 following:

```
80317 (cd /tmp)
80318 nohup cd
80319 find . -exec cd {} \;
```

80320 it does not affect the working directory of the caller's environment.

80321 The user must have execute (search) permission in *directory* in order to change to it.80322 **EXAMPLES**

80323 None.

80324 **RATIONALE**

80325 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of  
 80326 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

80327 A common extension when *HOME* is undefined is to get the login directory from the user  
 80328 database for the invoking user. This does not occur on System V implementations.

80329 Some historical shells, such as the KornShell, took special actions when the directory name  
 80330 contained a dot-dot component, selecting the logical parent of the directory, rather than the  
 80331 actual parent directory; that is, it moved up one level toward the '/' in the pathname,  
 80332 remembering what the user typed, rather than performing the equivalent of:

```
80333 chdir("../");
```

80334 In such a shell, the following commands would not necessarily produce equivalent output for all  
 80335 directories:

```
80336 cd .. && ls      ls ..
```

80337 This behavior is now the default. It is not consistent with the definition of dot-dot in most  
 80338 historical practice; that is, while this behavior has been optionally available in the KornShell,  
 80339 other shells have historically not supported this functionality. The logical pathname is stored in  
 80340 the *PWD* environment variable when the *cd* utility completes and this value is used to construct  
 80341 the next directory name if *cd* is invoked with the *-L* option.

80342 **FUTURE DIRECTIONS**

80343 None.

80344 **SEE ALSO**80345 [Section 2.12](#) (on page 2331), *pwd*80346 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)80347 XSH *chdir()*80348 **CHANGE HISTORY**

80349 First released in Issue 2.

80350 **Issue 6**80351 The following new requirements on POSIX implementations derive from alignment with the  
80352 Single UNIX Specification:

- 80353
- The *cd* – operand, *PWD*, and *OLDPWD* are added.

80354 The *-L* and *-P* options are added to align with the IEEE P1003.2b draft standard. This also  
80355 includes the introduction of a new description to include the effect of these options.80356 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to  
80357 make it clear that the *-L* and *-P* options are mutually-exclusive.80358 **Issue 7**

80359 Austin Group Interpretation 1003.1-2001 #037 is applied.

80360 Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *od* utility handles  
80361 concatenation of two pathnames when the first pathname ends in a <slash> character.

80362 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80363 Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”.80364 Changes to the *pwd* utility and *PWD* environment variable have been made to match the  
80365 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.

**cflow**

Utilities

80366 **NAME**80367 **cflow** — generate a C-language flowgraph (**DEVELOPMENT**)80368 **SYNOPSIS**

```
80369 xSI cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
80370 [-U dir]... file...
```

80371 **DESCRIPTION**

80372 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc*  
 80373 source files, and attempt to build a graph, written to standard output, charting the external  
 80374 references.

80375 **OPTIONS**

80376 The *cflow* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **-D**,  
 80377 **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant.

80378 The following options shall be supported:

- 80379 **-d** *num* Indicate the depth at which the flowgraph is cut off. The application shall ensure  
 80380 that the argument *num* is a decimal integer. By default this is a very large number  
 80381 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive  
 80382 integer shall be ignored.
- 80383 **-i** *incl* Increase the number of included symbols. The *incl* option-argument is one of the  
 80384 following characters:
- 80385 *x* Include external and static data symbols. The default shall be to include only  
 80386 functions in the flowgraph.
- 80387 *\_* (Underscore) Include names that begin with an <underscore>. The default  
 80388 shall be to exclude these functions (and data if **-i x** is used).
- 80389 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the  
 80390 callers of each function. The listing shall also be sorted in lexicographical order by  
 80391 callee.

80392 **OPERANDS**

80393 The following operand is supported:

- 80394 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by  
 80395 **.l** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the  
 80396 output of *c99* **-E**. Such files shall be processed as appropriate, determined by their  
 80397 suffix.
- 80398 Files suffixed by **.s** (conventionally assembler source) may have more limited  
 80399 information extracted from them.

80400 **STDIN**

80401 Not used.

80402 **INPUT FILES**

80403 The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

80404 **ENVIRONMENT VARIABLES**

80405 The following environment variables shall affect the execution of *cflow*:

- 80406 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 80407 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 80408 variables used to determine the values of locale categories.)

|       |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 80409 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 80410 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80411 | <i>LC_COLLATE</i>             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80412 |                               | Determine the locale for the ordering of the output when the <i>-r</i> option is used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80413 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 80414 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80415 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80416 | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80417 |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 80418 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80419 | <i>NLSPATH</i>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 80420 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80421 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 80422 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80423 |                               | The flowgraph written to standard output shall be formatted as follows:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 80424 |                               | "%d %s:%s\n", <reference number>, <global>, <definition>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 80425 |                               | Each line of output begins with a reference (that is, line) number, followed by indentation of at least one column position per level. This is followed by the name of the global, a <colon>, and its definition. Normally globals are only functions not defined as an external or beginning with an <underscore>; see the <b>OPTIONS</b> section for the <i>-i</i> inclusion option. For information extracted from C-language source, the definition consists of an abstract type declaration (for example, <b>char</b> *) and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the filename and location counter under which the symbol appeared (for example, <i>text</i> ). |
| 80426 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80427 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80428 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80429 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80430 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80431 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80432 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80433 |                               | Once a definition of a name has been written, subsequent references to that name contain only the reference number of the line where the definition can be found. For undefined references, only "<>" shall be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 80434 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80435 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80436 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80437 |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 80438 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80439 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 80440 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80441 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 80442 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80443 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 80444 |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 80445 |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 80446 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 80447 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

80448 **APPLICATION USAGE**

80449 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can  
 80450 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

80451 **EXAMPLES**

80452 Given the following in **file.c**:

```
80453     int i;
80454     int f();
80455     int g();
80456     int h();
80457     int
80458     main()
80459     {
80460         f();
80461         g();
80462         f();
80463     }
80464     int
80465     f()
80466     {
80467         i = h();
80468     }
```

80469 The command:

```
80470 cflow -i x file.c
```

80471 produces the output:

```
80472 1 main: int(), <file.c 6>
80473 2   f: int(), <file.c 13>
80474 3     h: <>
80475 4     i: int, <file.c 1>
80476 5     g: <>
```

80477 **RATIONALE**

80478 None.

80479 **FUTURE DIRECTIONS**

80480 None.

80481 **SEE ALSO**

80482 [c99](#), [lex](#), [yacc](#)

80483 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

80484 **CHANGE HISTORY**

80485 First released in Issue 2.

80486 **Issue 6**

80487 The normative text is reworded to avoid use of the term “must” for application requirements.

80488 **Issue 7**

80489 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80490 **NAME**

80491 chgrp — change the file group ownership

80492 **SYNOPSIS**80493 chgrp [-h] *group file...*80494 chgrp -R [-H|-L|-P] *group file...*80495 **DESCRIPTION**80496 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID  
80497 specified by the *group* operand.80498 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
80499 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to  
80500 the *chown()* function defined in the System Interfaces volume of POSIX.1-2008, called with the  
80501 following arguments:

- 80502 • The *file* operand shall be used as the *path* argument.
- 80503 • The user ID of the file shall be used as the *owner* argument.
- 80504 • The specified group ID shall be used as the *group* argument.

80505 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
80506 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
80507 group-ID bits of other file types may be cleared.80508 **OPTIONS**80509 The *chgrp* utility shall conform to XBD Section 12.2 (on page 215).

80510 The following options shall be supported by the implementation:

- 80511 **-h** If the system supports group IDs for symbolic links, for each *file* operand that  
80512 names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the  
80513 symbolic link instead of the file referenced by the symbolic link. If the system does  
80514 not support group IDs for symbolic links, for each *file* operand that names a file of  
80515 type symbolic link, *chgrp* shall do nothing more with the current file and shall go  
80516 on to any remaining files.
- 80517 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
80518 is specified on the command line, *chgrp* shall change the group of the directory  
80519 referenced by the symbolic link and all files in the file hierarchy below it.
- 80520 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
80521 is specified on the command line or encountered during the traversal of a file  
80522 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic  
80523 link and all files in the file hierarchy below it.
- 80524 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
80525 or encountered during the traversal of a file hierarchy, *chgrp* shall change the group  
80526 ID of the symbolic link if the system supports this operation. The *chgrp* utility shall  
80527 not follow the symbolic link to any other part of the file hierarchy.
- 80528 **-R** Recursively change file group IDs. For each *file* operand that names a directory,  
80529 *chgrp* shall change the group of the directory and all files in the file hierarchy  
80530 below it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these  
80531 options will be used as the default.

80532 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
80533 considered an error. The last option specified shall determine the behavior of the utility.

**chgrp**

Utilities

80534 **OPERANDS**

80535 The following operands shall be supported:

80536 *group* A group name from the group database or a numeric group ID. Either specifies a  
 80537 group ID to be given to each file named by one of the *file* operands. If a numeric  
 80538 *group* operand exists in the group database as a group name, the group ID number  
 80539 associated with that group name is used as the group ID.

80540 *file* A pathname of a file whose group ID is to be modified.

80541 **STDIN**

80542 Not used.

80543 **INPUT FILES**

80544 None.

80545 **ENVIRONMENT VARIABLES**80546 The following environment variables shall affect the execution of *chgrp*:

80547 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 80548 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 80549 variables used to determine the values of locale categories.)

80550 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 80551 internationalization variables.

80552 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 80553 characters (for example, single-byte as opposed to multi-byte characters in  
 80554 arguments).

80555 *LC\_MESSAGES*

80556 Determine the locale that should be used to affect the format and contents of  
 80557 diagnostic messages written to standard error.

80558 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

80559 **ASYNCHRONOUS EVENTS**

80560 Default.

80561 **STDOUT**

80562 Not used.

80563 **STDERR**

80564 The standard error shall be used only for diagnostic messages.

80565 **OUTPUT FILES**

80566 None.

80567 **EXTENDED DESCRIPTION**

80568 None.

80569 **EXIT STATUS**

80570 The following exit values shall be returned:

80571 0 The utility executed successfully and all requested changes were made.

80572 &gt;0 An error occurred.

80573 **CONSEQUENCES OF ERRORS**

80574 Default.

80575 **APPLICATION USAGE**80576 Only the owner of a file or the user with appropriate privileges may change the owner or group  
80577 of a file.80578 Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the  
80579 *group* specified is not the effective group ID or one of the supplementary group IDs of the calling  
80580 process.80581 **EXAMPLES**

80582 None.

80583 **RATIONALE**80584 The System V and BSD versions use different exit status codes. Some implementations used the  
80585 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
80586 can overflow the range of valid exit status values. The standard developers chose to mask these  
80587 by specifying only 0 and >0 as exit values.80588 The functionality of *chgrp* is described substantially through references to *chown*(). In this way,  
80589 there is no duplication of effort required for describing the interactions of permissions, multiple  
80590 groups, and so on.80591 **FUTURE DIRECTIONS**

80592 None.

80593 **SEE ALSO**80594 *chmod*, *chown*

80595 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

80596 XSH *chown*()80597 **CHANGE HISTORY**

80598 First released in Issue 2.

80599 **Issue 6**80600 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
80601 options affect the processing of symbolic links.80602 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
80603 section to "Default".80604 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to  
80605 make it clear that **-h** and **-R** are optional.80606 **Issue 7**80607 SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

80608 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**chmod**

Utilities

80609 **NAME**80610 `chmod` — change the file modes80611 **SYNOPSIS**80612 `chmod [-R] mode file...`80613 **DESCRIPTION**80614 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*  
80615 operand in the way specified by the *mode* operand.80616 It is implementation-defined whether and how the *chmod* utility affects any alternate or  
80617 additional file access control mechanism (see XBD Section 4.4, on page 108) being used for the  
80618 specified file.80619 Only a process whose effective user ID matches the user ID of the file, or a process with  
80620 appropriate privileges, shall be permitted to change the file mode bits of a file.80621 Upon successfully changing the file mode bits of a file, the *chmod* utility shall mark for update  
80622 the last file status change timestamp of the file.80623 **OPTIONS**80624 The *chmod* utility shall conform to XBD Section 12.2 (on page 215).

80625 The following option shall be supported:

80626 **-R** Recursively change file mode bits. For each *file* operand that names a directory,  
80627 *chmod* shall change the file mode bits of the directory and all files in the file  
80628 hierarchy below it.80629 **OPERANDS**

80630 The following operands shall be supported:

80631 *mode* Represents the change to be made to the file mode bits of each file named by one of  
80632 the *file* operands; see the EXTENDED DESCRIPTION section.80633 *file* A pathname of a file whose file mode bits shall be modified.80634 **STDIN**

80635 Not used.

80636 **INPUT FILES**

80637 None.

80638 **ENVIRONMENT VARIABLES**80639 The following environment variables shall affect the execution of *chmod*:80640 **LANG** Provide a default value for the internationalization variables that are unset or null.  
80641 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
80642 variables used to determine the values of locale categories.)80643 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
80644 internationalization variables.80645 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
80646 characters (for example, single-byte as opposed to multi-byte characters in  
80647 arguments).80648 **LC\_MESSAGES**80649 Determine the locale that should be used to affect the format and contents of  
80650 diagnostic messages written to standard error.

|       |     |                             |                                                                                                                                                        |
|-------|-----|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 80651 | XSI | <b>NLSPATH</b>              | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                  |
| 80652 |     | <b>ASYNCHRONOUS EVENTS</b>  |                                                                                                                                                        |
| 80653 |     |                             | Default.                                                                                                                                               |
| 80654 |     | <b>STDOUT</b>               |                                                                                                                                                        |
| 80655 |     |                             | Not used.                                                                                                                                              |
| 80656 |     | <b>STDERR</b>               |                                                                                                                                                        |
| 80657 |     |                             | The standard error shall be used only for diagnostic messages.                                                                                         |
| 80658 |     | <b>OUTPUT FILES</b>         |                                                                                                                                                        |
| 80659 |     |                             | None.                                                                                                                                                  |
| 80660 |     | <b>EXTENDED DESCRIPTION</b> |                                                                                                                                                        |
| 80661 |     |                             | The <i>mode</i> operand shall be either a <i>symbolic_mode</i> expression or a non-negative octal integer. The                                         |
| 80662 |     |                             | <i>symbolic_mode</i> form is described by the grammar later in this section.                                                                           |
| 80663 |     |                             | Each <b>clause</b> shall specify an operation to be performed on the current file mode bits of each <i>file</i> .                                      |
| 80664 |     |                             | The operations shall be performed on each <i>file</i> in the order in which the <b>clauses</b> are specified.                                          |
| 80665 |     |                             | The <b>who</b> symbols <b>u</b> , <b>g</b> , and <b>o</b> shall specify the <i>user</i> , <i>group</i> , and <i>other</i> parts of the file mode bits, |
| 80666 |     |                             | respectively. A <b>who</b> consisting of the symbol <b>a</b> shall be equivalent to <b>ugo</b> .                                                       |
| 80667 |     |                             | The <b>perm</b> symbols <b>r</b> , <b>w</b> , and <b>x</b> represent the <i>read</i> , <i>write</i> , and <i>execute/search</i> portions of file mode  |
| 80668 |     |                             | bits, respectively. The <b>perm</b> symbol <b>s</b> shall represent the <i>set-user-ID-on-execution</i> (when <b>who</b>                               |
| 80669 |     |                             | contains or implies <b>u</b> ) and <i>set-group-ID-on-execution</i> (when <b>who</b> contains or implies <b>g</b> ) bits.                              |
| 80670 |     |                             | The <b>perm</b> symbol <b>X</b> shall represent the <i>execute/search</i> portion of the file mode bits if the file is a                               |
| 80671 |     |                             | directory or if the current (unmodified) file mode bits have at least one of the execute bits                                                          |
| 80672 |     |                             | ( <b>S_IXUSR</b> , <b>S_IXGRP</b> , or <b>S_IXOTH</b> ) set. It shall be ignored if the file is not a directory and none of                            |
| 80673 |     |                             | the execute bits are set in the current file mode bits.                                                                                                |
| 80674 |     |                             | The <b>permcop</b> symbols <b>u</b> , <b>g</b> , and <b>o</b> shall represent the current permissions associated with the                              |
| 80675 |     |                             | <i>user</i> , <i>group</i> , and <i>other</i> parts of the file mode bits, respectively. For the remainder of this section,                            |
| 80676 |     |                             | <b>perm</b> refers to the non-terminals <b>perm</b> and <b>permcop</b> in the grammar.                                                                 |
| 80677 |     |                             | If multiple <b>actionlists</b> are grouped with a single <b>wholist</b> in the grammar, each <b>actionlist</b> shall be                                |
| 80678 |     |                             | applied in the order specified with that <b>wholist</b> . The <i>op</i> symbols shall represent the operation                                          |
| 80679 |     |                             | performed, as follows:                                                                                                                                 |
| 80680 |     | +                           | If <b>perm</b> is not specified, the '+' operation shall not change the file mode bits.                                                                |
| 80681 |     |                             | If <b>who</b> is not specified, the file mode bits represented by <b>perm</b> for the owner, group, and                                                |
| 80682 |     |                             | other permissions, except for those with corresponding bits in the file mode creation mask                                                             |
| 80683 |     |                             | of the invoking process, shall be set.                                                                                                                 |
| 80684 |     |                             | Otherwise, the file mode bits represented by the specified <b>who</b> and <b>perm</b> values shall be set.                                             |
| 80685 |     | -                           | If <b>perm</b> is not specified, the '-' operation shall not change the file mode bits.                                                                |
| 80686 |     |                             | If <b>who</b> is not specified, the file mode bits represented by <b>perm</b> for the owner, group, and                                                |
| 80687 |     |                             | other permissions, except for those with corresponding bits in the file mode creation mask                                                             |
| 80688 |     |                             | of the invoking process, shall be cleared.                                                                                                             |
| 80689 |     |                             | Otherwise, the file mode bits represented by the specified <b>who</b> and <b>perm</b> values shall be                                                  |
| 80690 |     |                             | cleared.                                                                                                                                               |
| 80691 |     | =                           | Clear the file mode bits specified by the <b>who</b> value, or, if no <b>who</b> value is specified, all of the                                        |
| 80692 |     |                             | file mode bits specified in this volume of POSIX.1-2008.                                                                                               |

**chmod**

80693 If **perm** is not specified, the '=' operation shall make no further modifications to the file  
80694 mode bits.

80695 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and  
80696 other permissions, except for those with corresponding bits in the file mode creation mask  
80697 of the invoking process, shall be set.

80698 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

80699 When using the symbolic mode form on a regular file, it is implementation-defined whether or  
80700 not:

- 80701 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all  
80702 execute bits are currently clear and none are being set are ignored.
- 80703 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-  
80704 on-execution bits.
- 80705 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all  
80706 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s*  
80707 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is  
80708 set, the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

80709 When using the symbolic mode form on other file types, it is implementation-defined whether  
80710 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
80711 honored.

80712 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols  
80713 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be  
80714 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**  
80715 symbol **s**.

80716 XSI The **perm** symbol **t** shall specify the S\_ISVTX bit. When used with a file of type directory, it can  
80717 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**  
80718 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these  
80719 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other  
80720 than directory is unspecified.

80721 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

80722 For each bit set in the octal number, the corresponding file permission bit shown in the following  
80723 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in  
80724 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-  
80725 execution, bits shown in the following table shall be set; if these bits are not set in the octal  
80726 number, they are cleared. For other file types, it is implementation-defined whether or not  
80727 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
80728 honored.

| Octal | Mode Bit | | | | | | |
|---|---|---|---|---|---|---|---|
| 4000  | S_ISUID  | 0400  | S_IRUSR  | 0040  | S_IRGRP  | 0004  | S_IROTH  |
| 2000  | S_ISGID  | 0200  | S_IWUSR  | 0020  | S_IWGRP  | 0002  | S_IWOTH  |
| 1000  | S_ISVTX  | 0100  | S_IXUSR  | 0010  | S_IXGRP  | 0001  | S_IXOTH  |

80729  
80730  
80731  
80732 XSI  
80733 When bits are set in the octal number other than those listed in the table above, the behavior is  
80734 unspecified.

80735 **Grammar for chmod**

80736 The grammar and lexical conventions in this section describe the syntax for the *symbolic\_mode*  
 80737 operand. The general conventions for this style of grammar are described in [Section 1.3](#) (on page  
 80738 2287). A valid *symbolic\_mode* can be represented as the non-terminal symbol *symbolic\_mode* in  
 80739 the grammar. This formal syntax shall take precedence over the preceding text syntax  
 80740 description.

80741 The lexical processing is based entirely on single characters. Implementations need not allow  
 80742 <blank> characters within the single argument being processed.

```
80743 %start    symbolic_mode
80744 %%
```

```
80745 symbolic_mode    : clause
80746                  | symbolic_mode ',' clause
80747                  ;
```

```
80748 clause          : actionlist
80749                  | wholist actionlist
80750                  ;
```

```
80751 wholist         : who
80752                  | wholist who
80753                  ;
```

```
80754 who            : 'u' | 'g' | 'o' | 'a'
80755                  ;
```

```
80756 actionlist     : action
80757                  | actionlist action
80758                  ;
```

```
80759 action         : op
80760                  | op permlist
80761                  | op permcopy
80762                  ;
```

```
80763 permcopy      : 'u' | 'g' | 'o'
80764                  ;
```

```
80765 op            : '+' | '-' | '='
80766                  ;
```

```
80767 permlist      : perm
80768                  | perm permlist
80769                  ;
```

```
80770 XSI perm      : 'r' | 'w' | 'x' | 'X' | 's' | 't'
80771                  ;
```

80772 **EXIT STATUS**

80773 The following exit values shall be returned:

80774 0 The utility executed successfully and all requested changes were made.

80775 >0 An error occurred.

# chmod

80776 **CONSEQUENCES OF ERRORS**

80777 Default.

80778 **APPLICATION USAGE**

80779 Some implementations of the *chmod* utility change the mode of a directory before the files in the  
 80780 directory when performing a recursive (**-R** option) change; others change the directory mode  
 80781 after the files in the directory. If an application tries to remove read or search permission for a  
 80782 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying  
 80783 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users  
 80784 should not try to make a hierarchy inaccessible to themselves.

80785 Some implementations of *chmod* never used the *umask* of the process when changing modes;  
 80786 systems conformant with this volume of POSIX.1-2008 do so when **who** is not specified. Note  
 80787 the difference between:

80788 `chmod a-w file`

80789 which removes all write permissions, and:

80790 `chmod -- -w file`

80791 which removes write permissions that would be allowed if **file** was created with the same  
 80792 *umask*.

80793 Conforming applications should never assume that they know how the set-user-ID and set-  
 80794 group-ID bits on directories are interpreted.

80795 **EXAMPLES**

| Mode         | Results                                                                                            |
|--------------|----------------------------------------------------------------------------------------------------|
| <i>a+=</i>   | Equivalent to <i>a+,a=</i> ; clears all file mode bits.                                            |
| <i>go+-w</i> | Equivalent to <i>go+,go-w</i> ; clears group and other write bits.                                 |
| <i>g=0-w</i> | Equivalent to <i>g=0,g-w</i> ; sets group bit to match other bits and then clears group write bit. |
| <i>g-r+w</i> | Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.                     |
| <i>u0=g</i>  | Sets owner bits to match group bits and sets other bits to match group bits.                       |

80806 **RATIONALE**

80807 The functionality of *chmod* is described substantially through references to concepts defined in  
 80808 the System Interfaces volume of POSIX.1-2008. In this way, there is less duplication of effort  
 80809 required for describing the interactions of permissions. However, the behavior of this utility is  
 80810 not described in terms of the *chmod()* function from the System Interfaces volume of  
 80811 POSIX.1-2008 because that specification requires certain side-effects upon alternate file access  
 80812 control mechanisms that might not be appropriate, depending on the implementation.

80813 Implementations that support mandatory file and record locking as specified by the 1984  
 80814 /usr/group standard historically used the combination of set-group-ID bit set and group  
 80815 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the  
 80816 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory  
 80817 locking mode is not changed without explicit indication that that was what the user intended.  
 80818 Therefore, the details on how the implementation treats these conditions must be defined in the  
 80819 documentation. This volume of POSIX.1-2008 does not require mandatory locking (nor does the  
 80820 System Interfaces volume of POSIX.1-2008), but does allow it as an extension. However, this  
 80821 volume of POSIX.1-2008 does require that the *ls* and *chmod* utilities work consistently in this

80822 area. If `ls -l file` indicates that the set-group-ID bit is set, `chmod g-s file` must clear it (assuming  
80823 appropriate privileges exist to change modes).

80824 The System V and BSD versions use different exit status codes. Some implementations used the  
80825 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
80826 can overflow the range of valid exit status values. This problem is avoided here by specifying  
80827 only 0 and >0 as exit values.

80828 The System Interfaces volume of POSIX.1-2008 indicates that implementation-defined  
80829 restrictions may cause the S\_ISUID and S\_ISGID bits to be ignored. This volume of POSIX.1-2008  
80830 allows the `chmod` utility to choose to modify these bits before calling `chmod()` (or some function  
80831 providing equivalent capabilities) for non-regular files. Among other things, this allows  
80832 implementations that use the set-user-ID and set-group-ID bits on directories to enable extended  
80833 features to handle these extensions in an intelligent manner.

80834 The **X perm** symbol was adopted from BSD-based systems because it provides commonly  
80835 desired functionality when doing recursive (`-R` option) modifications. Similar functionality is  
80836 not provided by the `find` utility. Historical BSD versions of `chmod`, however, only supported **X**  
80837 with `op+`; it has been extended in this volume of POSIX.1-2008 because it is also useful with `op=`.  
80838 (It has also been added for `op-` even though it duplicates `x`, in this case, because it is intuitive  
80839 and easier to explain.)

80840 The grammar was extended with the `permcoppy` non-terminal to allow historical-practice forms of  
80841 symbolic modes like `o=u-g` (that is, set the “other” permissions to the permissions of “owner”  
80842 minus the permissions of “group”).

#### 80843 FUTURE DIRECTIONS

80844 None.

#### 80845 SEE ALSO

80846 [ls](#), [umask](#)

80847 XBD [Section 4.4](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

80848 XSH [chmod\(\)](#)

#### 80849 CHANGE HISTORY

80850 First released in Issue 2.

#### 80851 Issue 6

80852 The following new requirements on POSIX implementations derive from alignment with the  
80853 Single UNIX Specification:

- 80854 • Octal modes have been kept and made mandatory despite being marked obsolescent in the  
80855 ISO POSIX-2: 1993 standard.

80856 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
80857 section to “Default.”.

80858 The Open Group Base Resolution bwg2001-010 is applied, adding the description of the  
80859 S\_ISVTX bit and the **t perm** symbol as part of the XSI option.

80860 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded  
80861 text in the EXTENDED DESCRIPTION from:

80862 “The **perm** symbol **t** shall specify the S\_ISVTX bit and shall apply to directories only. The  
80863 effect when using it with any other file type is unspecified. It can be used with the **who**  
80864 symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**  
80865 or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

**chmod***Utilities*

80866 to:

80867 “The **perm** symbol **t** shall specify the S\_ISVTX bit. When used with a file of type directory,  
80868 it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to  
80869 specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning  
80870 of these combinations is unspecified. The effect when using the **perm** symbol **t** with any  
80871 file type other than directory is unspecified.”

80872 This change is to permit historical behavior.

80873 **Issue 7**

80874 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80875 Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION  
80876 about about marking for update the last file status change timestamp of the file.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

80877 **NAME**

80878 chown — change the file ownership

80879 **SYNOPSIS**

80880 chown [-h] owner[:group] file...

80881 chown -R [-H|-L|-P] owner[:group] file...

80882 **DESCRIPTION**80883 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID  
80884 specified by the *owner* operand.80885 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
80886 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to  
80887 the *chown()* function defined in the System Interfaces volume of POSIX.1-2008, called with the  
80888 following arguments:

- 80889 1. The *file* operand shall be used as the *path* argument.
- 80890 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*  
80891 argument.
- 80892 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used  
80893 as the *group* argument; otherwise, the group ownership shall not be changed.

80894 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
80895 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
80896 group-ID bits of other file types may be cleared.80897 **OPTIONS**80898 The *chown* utility shall conform to XBD Section 12.2 (on page 215).

80899 The following options shall be supported by the implementation:

- 80900 **-h** For each file operand that names a file of type symbolic link, *chown* shall attempt to  
80901 set the user ID of the symbolic link. If a group ID was specified, for each file  
80902 operand that names a file of type symbolic link, *chown* shall attempt to set the  
80903 group ID of the symbolic link.
- 80904 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
80905 is specified on the command line, *chown* shall change the user ID (and group ID, if  
80906 specified) of the directory referenced by the symbolic link and all files in the file  
80907 hierarchy below it.
- 80908 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
80909 is specified on the command line or encountered during the traversal of a file  
80910 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the  
80911 directory referenced by the symbolic link and all files in the file hierarchy below it.
- 80912 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
80913 or encountered during the traversal of a file hierarchy, *chown* shall change the  
80914 owner ID (and group ID, if specified) of the symbolic link. The *chown* utility shall  
80915 not follow the symbolic link to any other part of the file hierarchy.
- 80916 **-R** Recursively change file user and group IDs. For each *file* operand that names a  
80917 directory, *chown* shall change the user ID (and group ID, if specified) of the  
80918 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is  
80919 specified, it is unspecified which of these options will be used as the default.

80920 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be

**chown**

Utilities

80921 considered an error. The last option specified shall determine the behavior of the utility.

80922 **OPERANDS**

80923 The following operands shall be supported:

80924 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this  
 80925 operand shall be a user name from the user database or a numeric user ID. Either  
 80926 specifies a user ID which shall be given to each file named by one of the *file*  
 80927 operands. If a numeric *owner* operand exists in the user database as a user name,  
 80928 the user ID number associated with that user name shall be used as the user ID.  
 80929 Similarly, if the *group* portion of this operand is present, it shall be a group name  
 80930 from the group database or a numeric group ID. Either specifies a group ID which  
 80931 shall be given to each file. If a numeric group operand exists in the group database  
 80932 as a group name, the group ID number associated with that group name shall be  
 80933 used as the group ID.

80934 *file* A pathname of a file whose user ID is to be modified.

80935 **STDIN**

80936 Not used.

80937 **INPUT FILES**

80938 None.

80939 **ENVIRONMENT VARIABLES**

80940 The following environment variables shall affect the execution of *chown*:

80941 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 80942 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 80943 variables used to determine the values of locale categories.)

80944 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 80945 internationalization variables.

80946 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 80947 characters (for example, single-byte as opposed to multi-byte characters in  
 80948 arguments).

80949 *LC\_MESSAGES*

80950 Determine the locale that should be used to affect the format and contents of  
 80951 diagnostic messages written to standard error.

80952 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

80953 **ASYNCHRONOUS EVENTS**

80954 Default.

80955 **STDOUT**

80956 Not used.

80957 **STDERR**

80958 The standard error shall be used only for diagnostic messages.

80959 **OUTPUT FILES**

80960 None.

80961 **EXTENDED DESCRIPTION**

80962 None.

80963 **EXIT STATUS**

80964 The following exit values shall be returned:

80965 0 The utility executed successfully and all requested changes were made.

80966 &gt;0 An error occurred.

80967 **CONSEQUENCES OF ERRORS**

80968 Default.

80969 **APPLICATION USAGE**80970 Only the owner of a file or the user with appropriate privileges may change the owner or group  
80971 of a file.80972 Some implementations restrict the use of *chown* to a user with appropriate privileges.80973 **EXAMPLES**

80974 None.

80975 **RATIONALE**80976 The System V and BSD versions use different exit status codes. Some implementations used the  
80977 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
80978 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0  
80979 as exit values.80980 The functionality of *chown* is described substantially through references to functions in the  
80981 System Interfaces volume of POSIX.1-2008. In this way, there is no duplication of effort required  
80982 for describing the interactions of permissions, multiple groups, and so on.80983 The 4.3 BSD method of specifying both owner and group was included in this volume of  
80984 POSIX.1-2008 because:

- 80985 • There are cases where the desired end condition could not be achieved using the *chgrp* and  
80986 *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the  
80987 desired group and the desired owner is not a member of the current group, the *chown*()  
80988 function could fail unless both owner and group are changed at the same time.)
- 80989 • Even if they could be changed independently, in cases where both are being changed, there  
80990 is a 100% performance penalty caused by being forced to invoke both utilities.

80991 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of POSIX.1-2008 because  
80992 the <period> is a valid character in login names (as specified by the Base Definitions volume of  
80993 POSIX.1-2008, login names consist of characters in the portable filename character set). The  
80994 <colon> character was chosen as the replacement for the <period> character because it would  
80995 never be allowed as a character in a user name or group name on historical implementations.

80996 The **-R** option is considered by some observers as an undesirable departure from the historical  
80997 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there  
80998 seemed to be no good reason to require other tools to have to duplicate that functionality.  
80999 However, the **-R** option was deemed an important user convenience, is far more efficient than  
81000 forking a separate process for each element of the directory hierarchy, and is in widespread  
81001 historical use.

**chown**

Utilities

81002 **FUTURE DIRECTIONS**

81003 None.

81004 **SEE ALSO**81005 *chgrp, chmod*

81006 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

81007 XSH *chown()*81008 **CHANGE HISTORY**

81009 First released in Issue 2.

81010 **Issue 6**81011 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

81012 The normative text is reworded to avoid use of the term “must” for application requirements.

81013 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to “Default.”.

81014 The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group ownership shall not be changed”.

81015 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to make it clear that **-h** and **-R** are optional.81020 **Issue 7**81021 SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS.

81022 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81023 The description of the **-h** and **-P** options is revised.

81024 **NAME**

81025 cksum — write file checksums and sizes

81026 **SYNOPSIS**81027 cksum [*file...*]81028 **DESCRIPTION**

81029 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)  
 81030 for each input file, and also write to standard output the number of octets in each file. The CRC  
 81031 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996  
 81032 standard (Ethernet).

81033 The encoding for the CRC checksum is defined by the generating polynomial:

$$81034 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

81035 Mathematically, the CRC value corresponding to a given file shall be defined by the following  
 81036 procedure:

- 81037 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial  
 81038  $M(x)$  of degree  $n-1$ . These *n* bits are the bits from the file, with the most significant bit  
 81039 being the most significant bit of the first octet of the file and the last bit being the least  
 81040 significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral  
 81041 number of octets, followed by one or more octets representing the length of the file as a  
 81042 binary value, least significant octet first. The smallest number of octets capable of  
 81043 representing this integer shall be used.
- 81044 2.  $M(x)$  is multiplied by  $x^{32}$  (that is, shifted left 32 bits) and divided by  $G(x)$  using mod 2  
 81045 division, producing a remainder  $R(x)$  of degree  $\leq 31$ .
- 81046 3. The coefficients of  $R(x)$  are considered to be a 32-bit sequence.
- 81047 4. The bit sequence is complemented and the result is the CRC.

81048 **OPTIONS**

81049 None.

81050 **OPERANDS**

81051 The following operand shall be supported:

81052 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard  
 81053 input shall be used.

81054 **STDIN**

81055 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 81056 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 81057 the standard input shall not be used. See the INPUT FILES section.

81058 **INPUT FILES**

81059 The input files can be any file type.

81060 **ENVIRONMENT VARIABLES**81061 The following environment variables shall affect the execution of *cksum*:

81062 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 81063 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 81064 variables used to determine the values of locale categories.)

81065 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 81066 internationalization variables.

**cksum**

Utilities

- 81067 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
81068 characters (for example, single-byte as opposed to multi-byte characters in  
81069 arguments).
- 81070 *LC\_MESSAGES*  
81071 Determine the locale that should be used to affect the format and contents of  
81072 diagnostic messages written to standard error.
- 81073 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 81074 **ASYNCHRONOUS EVENTS**  
81075 Default.
- 81076 **STDOUT**  
81077 For each file processed successfully, the *cksum* utility shall write in the following format:  
81078 "%u %d %s\n", <checksum>, <# of octets>, <pathname>  
81079 If no *file* operand was specified, the *pathname* and its leading <space> shall be omitted.
- 81080 **STDERR**  
81081 The standard error shall be used only for diagnostic messages.
- 81082 **OUTPUT FILES**  
81083 None.
- 81084 **EXTENDED DESCRIPTION**  
81085 None.
- 81086 **EXIT STATUS**  
81087 The following exit values shall be returned:  
81088 0 All files were processed successfully.  
81089 >0 An error occurred.
- 81090 **CONSEQUENCES OF ERRORS**  
81091 Default.
- 81092 **APPLICATION USAGE**  
81093 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of  
81094 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this  
81095 comparison cannot be considered cryptographically secure. The chances of a damaged file  
81096 producing the same CRC as the original are small; deliberate deception is difficult, but probably  
81097 not impossible.
- 81098 Although input files to *cksum* can be any type, the results need not be what would be expected  
81099 on character special device files or on file types not described by the System Interfaces volume of  
81100 POSIX.1-2008. Since this volume of POSIX.1-2008 does not specify the block size used when  
81101 doing input, checksums of character special files need not process all of the data in those files.
- 81102 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted  
81103 between two systems and undergoes any data transformation (such as changing little-endian  
81104 byte ordering to big-endian), identical CRC values cannot be expected. Implementations  
81105 performing such transformations may extend *cksum* to handle such situations.
- 81106 **EXAMPLES**  
81107 None.

81108 **RATIONALE**

81109 The following C-language program can be used as a model to describe the algorithm. It assumes  
 81110 that a **char** is one octet. It also assumes that the entire file is available for one pass through the  
 81111 function. This was done for simplicity in demonstrating the algorithm, rather than as an  
 81112 implementation model.

```

81113 static unsigned long crctab[] = {
81114 0x00000000,
81115 0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
81116 0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
81117 0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbbdb,
81118 0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
81119 0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
81120 0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
81121 0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
81122 0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
81123 0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
81124 0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
81125 0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
81126 0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f0, 0xe13ef6f4,
81127 0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
81128 0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
81129 0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16,
81130 0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
81131 0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dddb, 0x6f52c06c,
81132 0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
81133 0x53dc6066, 0x4d9b3063, 0x495a2ad4, 0x44190b0d, 0x40d816ba,
81134 0xaca5c697, 0xa864db20, 0xa527fd9, 0xa1e6e04e, 0xbfa1b04b,
81135 0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
81136 0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
81137 0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
81138 0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
81139 0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
81140 0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
81141 0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
81142 0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
81143 0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
81144 0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
81145 0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
81146 0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
81147 0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
81148 0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
81149 0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcdalf604, 0xc960ebb3,
81150 0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
81151 0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
81152 0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
81153 0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
81154 0x4e8ee645, 0x4a4ffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
81155 0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
81156 0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
81157 0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
81158 0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,

```

```

81159      0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
81160      0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
81161      0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
81162      0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
81163      0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
81164      0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
81165      0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
81166      };

81167      unsigned long memcrc(const unsigned char *b, size_t n)
81168      {
81169      /*  Input arguments:
81170      *   const char*   b == byte sequence to checksum
81171      *   size_t       n == length of sequence
81172      */

81173          register unsigned   i, c, s = 0;

81174          for (i = n; i > 0; --i) {
81175              c = (unsigned) (*b++);
81176              s = (s << 8) ^ crctab[(s >> 24) ^ c];
81177          }

81178          /* Extend with the length of the string. */
81179          while (n != 0) {
81180              c = n & 0377;
81181              n >>= 8;
81182              s = (s << 8) ^ crctab[(s >> 24) ^ c];
81183          }

81184          return ~s;
81185      }

```

81186 The historical practice of writing the number of “blocks” has been changed to writing the  
81187 number of octets, since the latter is not only more useful, but also since historical  
81188 implementations have not been consistent in defining what a “block” meant.

81189 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the  
81190 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was  
81191 the default behavior on those systems, no realistic compromise was available if either were  
81192 selected—some set of historical applications would break. Therefore, the name was changed to  
81193 *cksum*. Although the historical *sum* commands will probably continue to be provided for many  
81194 years, programs designed for portability across systems should use the new name.

81195 The algorithm selected is based on that used by the ISO/IEC 8802-3: 1996 standard (Ethernet) for  
81196 the frame check sequence field. The algorithm used does not match the technical definition of a  
81197 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC  
81198 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also  
81199 because it guards against inadvertent collisions between files that begin with different series of  
81200 zero octets. The chance that two different files produce identical CRCs is much greater when  
81201 their lengths are not considered. Keeping the length and the checksum of the file itself separate  
81202 would yield a slightly more robust algorithm, but historical usage has always been that a single  
81203 number (the checksum as printed) represents the signature of the file. It was decided that  
81204 historical usage was the more important consideration.

81205 Early proposals contained modifications to the Ethernet algorithm that involved extracting table

81206 values whenever an intermediate result became zero. This was demonstrated to be less robust  
81207 than the current method and mathematically difficult to describe or justify.

81208 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.  
81209 The pseudo-code rendition is:

```
81210 X ← 0; Y ← 0;
81211 for i ← m - 1 step -1 until 0 do
81212     begin
81213         T ← X(1) ^ A[i];
81214         X(1) ← X(0); X(0) ← Y(1); Y(1) ← Y(0); Y(0) ← 0;
81215         comment: f[T] and f'[T] denote the T-th words in the
81216                 table f and f' ;
81217         X ← X ^ f[T]; Y ← Y ^ f'[T];
81218     end
```

81219 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**  
81220 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables  
81221 **f** and **f'** are a single table containing 32-bit values.

81222 The referenced Sarwate article also discusses generating the table.

#### 81223 FUTURE DIRECTIONS

81224 None.

#### 81225 SEE ALSO

81226 XBD [Chapter 8](#) (on page 173)

#### 81227 CHANGE HISTORY

81228 First released in Issue 4.

#### 81229 Issue 7

81230 Austin Group Interpretation 1003.1-2001 #092 is applied.

81231 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**cmp**

Utilities

81232 **NAME**81233 `cmp` — compare two files81234 **SYNOPSIS**81235 `cmp [-l|-s] file1 file2`81236 **DESCRIPTION**

81237 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the  
 81238 same. Under default options, if they differ, it shall write to standard output the byte and line  
 81239 number at which the first difference occurred. Bytes and lines shall be numbered beginning with  
 81240 1.

81241 **OPTIONS**81242 The *cmp* utility shall conform to XBD [Section 12.2](#) (on page 215).

81243 The following options shall be supported:

81244 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for  
 81245 each difference.

81246 **-s** Write nothing for differing files; return exit status only.

81247 **OPERANDS**

81248 The following operands shall be supported:

81249 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
 81250 be used.

81251 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input  
 81252 shall be used.

81253 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or  
 81254 character special file, the results are undefined.

81255 **STDIN**

81256 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the  
 81257 INPUT FILES section.

81258 **INPUT FILES**

81259 The input files can be any file type.

81260 **ENVIRONMENT VARIABLES**81261 The following environment variables shall affect the execution of *cmp*:

81262 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 81263 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 81264 variables used to determine the values of locale categories.)

81265 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 81266 internationalization variables.

81267 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 81268 characters (for example, single-byte as opposed to multi-byte characters in  
 81269 arguments).

81270 **LC\_MESSAGES**

81271 Determine the locale that should be used to affect the format and contents of  
 81272 diagnostic messages written to standard error and informative messages written to  
 81273 standard output.

- 81274 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 81275 **ASYNCHRONOUS EVENTS**
- 81276 Default.
- 81277 **STDOUT**
- 81278 In the POSIX locale, results of the comparison shall be written to standard output. When no
- 81279 options are used, the format shall be:
- 81280 "%s %s differ: char %d, line %d\n", *file1*, *file2*,
- 81281 <*byte number*>, <*line number*>
- 81282 When the **-l** option is used, the format shall be:
- 81283 "%d %o %o\n", <*byte number*>, <*differing byte*>,
- 81284 <*differing byte*>
- 81285 for each byte that differs. The first <*differing byte*> number is from *file1* while the second is from
- 81286 *file2*. In both cases, <*byte number*> shall be relative to the beginning of the file, beginning with 1.
- 81287 No output shall be written to standard output when the **-s** option is used.
- 81288 **STDERR**
- 81289 The standard error shall be used only for diagnostic messages. If the **-l** option is used and *file1*
- 81290 and *file2* differ in length, or if the **-s** option is not used and *file1* and *file2* are identical for the
- 81291 entire length of the shorter file, in the POSIX locale the following diagnostic message shall be
- 81292 written:
- 81293 "cmp: EOF on %s%s\n", <*name of shorter file*>, <*additional info*>
- 81294 The <*additional info*> field shall either be null or a string that starts with a <blank> and contains
- 81295 no <newline> characters. Some implementations report on the number of lines in this case.
- 81296 **OUTPUT FILES**
- 81297 None.
- 81298 **EXTENDED DESCRIPTION**
- 81299 None.
- 81300 **EXIT STATUS**
- 81301 The following exit values shall be returned:
- 81302 0 The files are identical.
- 81303 1 The files are different; this includes the case where one file is identical to the first part of the
- 81304 other.
- 81305 >1 An error occurred.
- 81306 **CONSEQUENCES OF ERRORS**
- 81307 Default.

81308 **APPLICATION USAGE**

81309 Although input files to *cmp* can be any type, the results might not be what would be expected on  
 81310 character special device files or on file types not described by the System Interfaces volume of  
 81311 POSIX.1-2008. Since this volume of POSIX.1-2008 does not specify the block size used when  
 81312 doing input, comparisons of character special files need not compare all of the data in those files.

81313 For files which are not text files, line numbers simply reflect the presence of a <newline>  
 81314 without any implication that the file is organized into lines.

81315 **EXAMPLES**

81316 None.

81317 **RATIONALE**

81318 The global language in [Section 1.4](#) (on page 2288) indicates that using two mutually-exclusive  
 81319 options together produces unspecified results. Some System V implementations consider the  
 81320 option usage:

81321 `cmp -l -s ...`

81322 to be an error. They also treat:

81323 `cmp -s -l ...`

81324 as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

81325 The word **char** in the standard output format comes from historical usage, even though it is  
 81326 actually a byte number. When *cmp* is supported in other locales, implementations are  
 81327 encouraged to use the word *byte* or its equivalent in another language. Users should not  
 81328 interpret this difference to indicate that the functionality of the utility changed between locales.

81329 Some implementations report on the number of lines in the identical-but-shorter file case. This is  
 81330 allowed by the inclusion of the <additional info> fields in the output format. The restriction on  
 81331 having a leading <blank> and no <newline> characters is to make parsing for the filename  
 81332 easier. It is recognized that some filenames containing white-space characters make parsing  
 81333 difficult anyway, but the restriction does aid programs used on systems where the names are  
 81334 predominantly well behaved.

81335 **FUTURE DIRECTIONS**

81336 None.

81337 **SEE ALSO**

81338 *comm*, *diff*

81339 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

81340 **CHANGE HISTORY**

81341 First released in Issue 2.

81342 **Issue 7**

81343 SD5-XCU-ERN-96 is applied, updating the STDERR section.

81344 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81345 **NAME**

81346 comm — select or reject lines common to two files

81347 **SYNOPSIS**81348 comm [-123] *file1 file2*81349 **DESCRIPTION**81350 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating  
81351 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and  
81352 lines in both files.81353 If the lines in both files are not ordered according to the collating sequence of the current locale,  
81354 the results are unspecified.81355 **OPTIONS**81356 The *comm* utility shall conform to XBD Section 12.2 (on page 215).

81357 The following options shall be supported:

- 81358 -1 Suppress the output column of lines unique to *file1*.
- 81359 -2 Suppress the output column of lines unique to *file2*.
- 81360 -3 Suppress the output column of lines duplicated in *file1* and *file2*.

81361 **OPERANDS**

81362 The following operands shall be supported:

- 81363 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
81364 be used.
- 81365 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input  
81366 shall be used.

81367 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or  
81368 character special file, the results are undefined.81369 **STDIN**81370 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.  
81371 See the INPUT FILES section.81372 **INPUT FILES**

81373 The input files shall be text files.

81374 **ENVIRONMENT VARIABLES**81375 The following environment variables shall affect the execution of *comm*:81376 *LANG* Provide a default value for the internationalization variables that are unset or null.  
81377 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
81378 variables used to determine the values of locale categories.)81379 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
81380 internationalization variables.81381 *LC\_COLLATE*81382 Determine the locale for the collating sequence *comm* expects to have been used  
81383 when the input files were sorted.81384 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
81385 characters (for example, single-byte as opposed to multi-byte characters in  
81386 arguments and input files).

- 81387 *LC\_MESSAGES*
- 81388 Determine the locale that should be used to affect the format and contents of
- 81389 diagnostic messages written to standard error.
- 81390 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 81391 **ASYNCHRONOUS EVENTS**
- 81392 Default.
- 81393 **STDOUT**
- 81394 The *comm* utility shall produce output depending on the options selected. If the *-1*, *-2*, and *-3*
- 81395 options are all selected, *comm* shall write nothing to standard output.
- 81396 If the *-1* option is not selected, lines contained only in *file1* shall be written using the format:
- 81397 "%s\n", <line in file1>
- 81398 If the *-2* option is not selected, lines contained only in *file2* are written using the format:
- 81399 "%s%s\n", <lead>, <line in file2>
- 81400 where the string <lead> is as follows:
- 81401 <tab> The *-1* option is not selected.
- 81402 null string The *-1* option is selected.
- 81403 If the *-3* option is not selected, lines contained in both files shall be written using the format:
- 81404 "%s%s\n", <lead>, <line in both>
- 81405 where the string <lead> is as follows:
- 81406 <tab><tab> Neither the *-1* nor the *-2* option is selected.
- 81407 <tab> Exactly one of the *-1* and *-2* options is selected.
- 81408 null string Both the *-1* and *-2* options are selected.
- 81409 If the input files were ordered according to the collating sequence of the current locale, the lines
- 81410 written shall be in the collating sequence of the original lines.
- 81411 **STDERR**
- 81412 The standard error shall be used only for diagnostic messages.
- 81413 **OUTPUT FILES**
- 81414 None.
- 81415 **EXTENDED DESCRIPTION**
- 81416 None.
- 81417 **EXIT STATUS**
- 81418 The following exit values shall be returned:
- 81419 0 All input files were successfully output as specified.
- 81420 >0 An error occurred.
- 81421 **CONSEQUENCES OF ERRORS**
- 81422 Default.

81423 **APPLICATION USAGE**

81424 If the input files are not properly presorted, the output of *comm* might not be useful.

81425 **EXAMPLES**

81426 If a file named **xcu** contains a sorted list of the utilities in this volume of POSIX.1-2008, a file  
 81427 named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue  
 81428 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface  
 81429 Definition Third Edition:

81430 `comm -23 xcu xpg3 | comm -23 - svid89`

81431 would print a list of utilities in this volume of POSIX.1-2008 not specified by either of the other  
 81432 documents:

81433 `comm -12 xcu xpg3 | comm -12 - svid89`

81434 would print a list of utilities specified by all three documents, and:

81435 `comm -12 xpg3 svid89 | comm -23 - xcu`

81436 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this  
 81437 volume of POSIX.1-2008.

81438 **RATIONALE**

81439 None.

81440 **FUTURE DIRECTIONS**

81441 None.

81442 **SEE ALSO**

81443 *cmp*, *diff*, *sort*, *uniq*

81444 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

81445 **CHANGE HISTORY**

81446 First released in Issue 2.

81447 **Issue 6**

81448 The normative text is reworded to avoid use of the term “must” for application requirements.

**command**

Utilities

81449 **NAME**81450 `command` — execute a simple command81451 **SYNOPSIS**81452 `command [-p] command_name [argument...]`81453 `command [-p] [-v|-V] command_name`81454 **DESCRIPTION**81455 The *command* utility shall cause the shell to treat the arguments as a simple command,  
81456 suppressing the shell function lookup that is described in Section 2.9.1.1 (on page 2317), item 1b.81457 If the *command\_name* is the same as the name of one of the special built-in utilities, the special  
81458 properties in the enumerated list at the beginning of Section 2.14 (on page 2334) shall not occur.  
81459 In every other respect, if *command\_name* is not the name of a function, the effect of *command*  
81460 (with no options) shall be the same as omitting *command*.81461 When the `-v` or `-V` option is used, the *command* utility shall provide information concerning  
81462 how a command name is interpreted by the shell.81463 **OPTIONS**81464 The *command* utility shall conform to XBD Section 12.2 (on page 215).

81465 The following options shall be supported:

81466 `-p` Perform the command search using a default value for *PATH* that is guaranteed to  
81467 find all of the standard utilities.81468 `-v` Write a string to standard output that indicates the pathname or command that  
81469 will be used by the shell, in the current shell execution environment (see Section  
81470 2.12, on page 2331), to invoke *command\_name*, but do not invoke *command\_name*.81471 • Utilities, regular built-in utilities, *command\_names* including a `<slash>`  
81472 character, and any implementation-defined functions that are found using  
81473 the *PATH* variable (as described in Section 2.9.1.1, on page 2317), shall be  
81474 written as absolute pathnames.81475 • Shell functions, special built-in utilities, regular built-in utilities not  
81476 associated with a *PATH* search, and shell reserved words shall be written as  
81477 just their names.81478 • An alias shall be written as a command line that represents its alias  
81479 definition.81480 • Otherwise, no output shall be written and the exit status shall reflect that the  
81481 name was not found.81482 `-V` Write a string to standard output that indicates how the name given in the  
81483 *command\_name* operand will be interpreted by the shell, in the current shell  
81484 execution environment (see Section 2.12, on page 2331), but do not invoke  
81485 *command\_name*. Although the format of this string is unspecified, it shall indicate  
81486 in which of the following categories *command\_name* falls and shall include the  
81487 information stated:81488 • Utilities, regular built-in utilities, and any implementation-defined functions  
81489 that are found using the *PATH* variable (as described in Section 2.9.1.1, on  
81490 page 2317), shall be identified as such and include the absolute pathname in  
81491 the string.

- 81492 • Other shell functions shall be identified as functions.
- 81493 • Aliases shall be identified as aliases and their definitions included in the
- 81494 string.
- 81495 • Special built-in utilities shall be identified as special built-in utilities.
- 81496 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 81497 as regular built-in utilities. (The term “regular” need not be used.)
- 81498 • Shell reserved words shall be identified as reserved words.

**81499 OPERANDS**

81500 The following operands shall be supported:

81501 *argument* One of the strings treated as an argument to *command\_name*.

81502 *command\_name*

81503 The name of a utility or a special built-in utility.

**81504 STDIN**

81505 Not used.

**81506 INPUT FILES**

81507 None.

**81508 ENVIRONMENT VARIABLES**

81509 The following environment variables shall affect the execution of *command*:

81510 *LANG* Provide a default value for the internationalization variables that are unset or null.  
81511 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
81512 variables used to determine the values of locale categories.)

81513 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
81514 internationalization variables.

81515 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
81516 characters (for example, single-byte as opposed to multi-byte characters in  
81517 arguments).

81518 *LC\_MESSAGES*

81519 Determine the locale that should be used to affect the format and contents of  
81520 diagnostic messages written to standard error and informative messages written to  
81521 standard output.

81522 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

81523 *PATH* Determine the search path used during the command search described in Section  
81524 2.9.1.1 (on page 2317), except as described under the *-p* option.

**81525 ASYNCHRONOUS EVENTS**

81526 Default.

**81527 STDOUT**

81528 When the *-v* option is specified, standard output shall be formatted as:

81529 "%s\n", *<pathname or command>*

81530 When the *-V* option is specified, standard output shall be formatted as:

81531 "%s\n", *<unspecified>*

**command**

Utilities

81532 **STDERR**

81533 The standard error shall be used only for diagnostic messages.

81534 **OUTPUT FILES**

81535 None.

81536 **EXTENDED DESCRIPTION**

81537 None.

81538 **EXIT STATUS**81539 When the `-v` or `-V` options are specified, the following exit values shall be returned:

81540 0 Successful completion.

81541 >0 The *command\_name* could not be found or an error occurred.

81542 Otherwise, the following exit values shall be returned:

81543 126 The utility specified by *command\_name* was found but could not be invoked.81544 127 An error occurred in the *command* utility or the utility specified by *command\_name* could not be found.81546 Otherwise, the exit status of *command* shall be that of the simple command specified by the arguments to *command*.81548 **CONSEQUENCES OF ERRORS**

81549 Default.

81550 **APPLICATION USAGE**81551 The order for command search allows functions to override regular built-ins and path searches.  
81552 This utility is necessary to allow functions that have the same name as a utility to call the utility  
81553 (instead of a recursive call to the function).81554 The system default path is available using *getconf*; however, since *getconf* may need to have the  
81555 *PATH* set up before it can be called itself, the following can be used:81556 `command -p getconf PATH`81557 There are some advantages to suppressing the special characteristics of special built-ins on  
81558 occasion. For example:81559 `command exec > unwritable-file`81560 does not cause a non-interactive script to abort, so that the output status can be checked by the  
81561 script.81562 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an  
81563 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility  
81564 exited with an error indication”. The value 127 was chosen because it is not commonly used for  
81565 other meanings; most utilities use small values for “normal error conditions” and the values  
81566 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen  
81567 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts  
81568 produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
81569 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
81570 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
81571 any other reason.81572 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution  
81573 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell  
81574 or separate utility execution environment, such as one of the following:

81575 (PATH=foo command -v)  
81576 nohup command -v

81577 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*  
81578 function, in a separate utility execution environment, most implementations are not able to  
81579 identify aliases, functions, or special built-ins.

81580 Two types of regular built-ins could be encountered on a system and these are described  
81581 separately by *command*. The description of command search in Section 2.9.1.1 (on page 2317)  
81582 allows for a standard utility to be implemented as a regular built-in as long as it is found in the  
81583 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or  
81584 some similar pathname. Other implementation-defined utilities that are not defined by this  
81585 volume of POSIX.1-2008 might exist only as built-ins and have no pathname associated with  
81586 them. These produce output identified as (regular) built-ins. Applications encountering these are  
81587 not able to count on *execing* them, using them with *nohup*, overriding them with a different  
81588 *PATH*, and so on.

#### 81589 EXAMPLES

- 81590 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
81591 cd() {
81592     command cd "$@" >/dev/null
81593     pwd
81594 }
```

- 81595 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
81596 IFS='
81597 '
81598 # The preceding value should be <space><tab><newline>.
81599 # Set IFS to its default value.

81600 \unalias -a
81601 # Unset all possible aliases.
81602 # Note that unalias is escaped to prevent an alias
81603 # being used for unalias.

81604 unset -f command
81605 # Ensure command is not a user function.

81606 PATH="$$(command -p getconf PATH):$PATH"
81607 # Put on a reliable PATH prefix.

81608 # ...
```

81609 At this point, given correct permissions on the directories called by *PATH*, the script has  
81610 the ability to ensure that any utility it calls is the intended one. It is being very cautious  
81611 because it assumes that implementation extensions may be present that would allow user  
81612 functions to exist when it is invoked; this capability is not specified by this volume of  
81613 POSIX.1-2008, but it is not prohibited as an extension. For example, the *ENV* variable  
81614 precedes the invocation of the script with a user start-up script. Such a script could define  
81615 functions to spoof the application.

#### 81616 RATIONALE

81617 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

81618 There is nothing in the description of *command* that implies the command line is parsed any  
81619 differently from that of any other simple command. For example:

81620 `command a | b ; c`

81621 is not parsed in any special way that causes `'|'` or  `';'`  to be treated other than a pipe operator  
81622 or `<semicolon>` or that prevents function lookup on `b` or `c`.

81623 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since  
81624 *command* also goes to the file system to search for utilities, the name *builtin* would not be  
81625 intuitive.

81626 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special  
81627 built-in for the following reasons:

- 81628 • The removal of exportable functions made the special precedence of a special built-in  
81629 unnecessary.
- 81630 • A special built-in has special properties (see [Section 2.14](#), on page 2334) that were  
81631 inappropriate for invoking other utilities. For example, two commands such as:

81632 `date > unwritable-file`

81633 `command date > unwritable-file`

81634 would have entirely different results; in a non-interactive script, the former would  
81635 continue to execute the next command, the latter would abort. Introducing this semantic  
81636 difference along with suppressing functions was seen to be non-intuitive.

81637 The `-p` option is present because it is useful to be able to ensure a safe path search that finds all  
81638 the standard utilities. This search might not be identical to the one that occurs through one of the  
81639 *exec* functions (as defined in the System Interfaces volume of POSIX.1-2008) when *PATH* is unset.  
81640 At the very least, this feature is required to allow the script to access the correct version of *getconf*  
81641 so that the value of the default path can be accurately retrieved.

81642 The *command* `-v` and `-V` options were added to satisfy requirements from users that are  
81643 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in  
81644 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what  
81645 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left  
81646 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more  
81647 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,  
81648 exported aliases, and undefined functions.

81649 The output format of `-V` was left mostly unspecified because human users are its only audience.  
81650 Applications should not be written to care about this information; they can use the output of `-v`  
81651 to differentiate between various types of commands, but the additional information that may be  
81652 emitted by the more verbose `-V` is not needed and should not be arbitrarily constrained in its  
81653 verbosity or localization for application parsing reasons.

#### 81654 FUTURE DIRECTIONS

81655 None.

#### 81656 SEE ALSO

81657 [Section 2.9.1.1](#) (on page 2317), [Section 2.12](#) (on page 2331), [Section 2.14](#) (on page 2334), *sh*, *type*

81658 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

81659 XSH *exec*

81660 **CHANGE HISTORY**

81661 First released in Issue 4.

81662 **Issue 7**

81663 Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow `-p` to  
81664 be used with `-v` (or `-V`).

81665 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81666 The *command* utility is moved from the User Portability Utilities option to the Base. User  
81667 Portability Utilities is now an option for interactive utilities.

81668 The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard  
81669 `getconf_CS_PATH` with `getconf PATH`.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**compress**

Utilities

81670 **NAME**81671 `compress` — compress data81672 **SYNOPSIS**81673 XSI `compress [-fv] [-b bits] [file...]`81674 `compress [-cfv] [-b bits] [file]`81675 **DESCRIPTION**81676 The *compress* utility shall attempt to reduce the size of the named files by using adaptive Lempel-  
81677 Ziv coding algorithm.81678 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,  
81679 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.81680 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on  
81681 December 10th, 1985, and assigned to Sperry Corporation.81682 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be  
81683 changed and an error value greater than two shall be returned. Except when the output is to the  
81684 standard output, each file shall be replaced by one with the extension *.Z*. If the invoking process  
81685 has appropriate privileges, the ownership, modes, access time, and modification time of the  
81686 original file are preserved. If appending the *.Z* to the filename would make the name exceed  
81687 {NAME\_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be  
81688 compressed to the standard output.81689 **OPTIONS**81690 The *compress* utility shall conform to XBD Section 12.2 (on page 215).

81691 The following options shall be supported:

81692 **-b** *bits* Specify the maximum number of bits to use in a code. For a conforming  
81693 application, the *bits* argument shall be:81694  $9 \leq bits \leq 14$ 81695 The implementation may allow *bits* values of greater than 14. The default is 14, 15,  
81696 or 16.81697 **-c** Cause *compress* to write to the standard output; the input file is not changed, and  
81698 no *.Z* files are created.81699 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if  
81700 the corresponding *file.Z* file already exists. If the **-f** option is not given, and the  
81701 process is not running in the background, the user is prompted as to whether an  
81702 existing *file.Z* file should be overwritten. If the response is affirmative, the existing  
81703 file will be overwritten.81704 **-v** Write the percentage reduction of each file to standard error.81705 **OPERANDS**

81706 The following operand shall be supported:

81707 *file* A pathname of a file to be compressed.81708 **STDIN**81709 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.

81710 **INPUT FILES**

81711 If *file* operands are specified, the input files contain the data to be compressed.

81712 **ENVIRONMENT VARIABLES**

81713 The following environment variables shall affect the execution of *compress*:

81714 *LANG* Provide a default value for the internationalization variables that are unset or null.  
81715 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
81716 variables used to determine the values of locale categories.)

81717 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
81718 internationalization variables.

81719 *LC\_COLLATE*

81720 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
81721 character collating elements used in the extended regular expression defined for  
81722 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

81723 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
81724 characters (for example, single-byte as opposed to multi-byte characters in  
81725 arguments), the behavior of character classes used in the extended regular  
81726 expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

81727 *LC\_MESSAGES*

81728 Determine the locale used to process affirmative responses, and the locale used to  
81729 affect the format and contents of diagnostic messages, prompts, and the output  
81730 from the **-v** option written to standard error.

81731 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

81732 **ASYNCHRONOUS EVENTS**

81733 Default.

81734 **STDOUT**

81735 If no *file* operands are specified, or if a *file* operand is '-', or if the **-c** option is specified, the  
81736 standard output contains the compressed output.

81737 **STDERR**

81738 The standard error shall be used only for diagnostic and prompt messages and the output from  
81739 **-v**.

81740 **OUTPUT FILES**

81741 The output files shall contain the compressed output. The format of compressed files is  
81742 unspecified and interchange of such files between implementations (including access via  
81743 unspecified file sharing mechanisms) is not required by POSIX.1-2008.

81744 **EXTENDED DESCRIPTION**

81745 None.

81746 **EXIT STATUS**

81747 The following exit values shall be returned:

81748 0 Successful completion.

81749 1 An error occurred.

81750 2 One or more files were not compressed because they would have increased in size (and the  
81751 **-f** option was not specified).

**compress**

Utilities

81752 >2 An error occurred.

81753 **CONSEQUENCES OF ERRORS**

81754 The input file shall remain unmodified.

81755 **APPLICATION USAGE**

81756 The amount of compression obtained depends on the size of the input, the number of *bits* per  
81757 code, and the distribution of common substrings. Typically, text such as source code or English is  
81758 reduced by 50-60%. Compression is generally much better than that achieved by Huffman  
81759 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

81760 Although *compress* strictly follows the default actions upon receipt of a signal or when an error  
81761 occurs, some unexpected results may occur. In some implementations it is likely that a partially  
81762 compressed file is left in place, alongside its uncompressed input file. Since the general  
81763 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been  
81764 successfully filled, an application should always carefully check the exit status of *compress* before  
81765 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

81766 The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the  
81767 restrictions imposed by the lack of an explicit published file format). Some implementations  
81768 based on 16-bit architectures cannot support 15 or 16-bit uncompression.

81769 **EXAMPLES**

81770 None.

81771 **RATIONALE**

81772 None.

81773 **FUTURE DIRECTIONS**

81774 None.

81775 **SEE ALSO**

81776 *uncompress*, *zcat*

81777 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

81778 **CHANGE HISTORY**

81779 First released in Issue 4

81780 **Issue 6**

81781 The normative text is reworded to avoid use of the term “must” for application requirements.

81782 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

81783 **Issue 7**

81784 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81785 Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT  
81786 VARIABLES section.

81787 **NAME**

81788 cp — copy files

81789 **SYNOPSIS**81790 cp [-PfiP] *source\_file target\_file*81791 cp [-PfiP] *source\_file... target*81792 cp -R [-H|-L|-P] [-fiP] *source\_file... target*81793 **DESCRIPTION**

81794 The first synopsis form is denoted by two operands, neither of which are existing files of type  
 81795 directory. The *cp* utility shall copy the contents of *source\_file* (or, if *source\_file* is a file of type  
 81796 symbolic link, the contents of the file referenced by *source\_file*) to the destination path named by  
 81797 *target\_file*.

81798 The second synopsis form is denoted by two or more operands where the **-R** option is not  
 81799 specified and the first synopsis form is not applicable. It shall be an error if any *source\_file* is a file  
 81800 of type directory, if *target* does not exist, or if *target* does not name a directory. The *cp* utility shall  
 81801 copy the contents of each *source\_file* (or, if *source\_file* is a file of type symbolic link, the contents of  
 81802 the file referenced by *source\_file*) to the destination path named by the concatenation of *target*, a  
 81803 single <slash> character if *target* did not end in a <slash>, and the last component of *source\_file*.

81804 The third synopsis form is denoted by two or more operands where the **-R** option is specified.  
 81805 The *cp* utility shall copy each file in the file hierarchy rooted in each *source\_file* to a destination  
 81806 path named as follows:

- 81807 • If *target* exists and names an existing directory, the name of the corresponding destination  
 81808 path for each file in the file hierarchy shall be the concatenation of *target*, a single <slash>  
 81809 character if *target* did not end in a <slash>, and the pathname of the file relative to the  
 81810 directory containing *source\_file*.
- 81811 • If *target* does not exist and two operands are specified, the name of the corresponding  
 81812 destination path for *source\_file* shall be *target*; the name of the corresponding destination  
 81813 path for all other files in the file hierarchy shall be the concatenation of *target*, a <slash>  
 81814 character, and the pathname of the file relative to *source\_file*.

81815 It shall be an error if *target* does not exist and more than two operands are specified, or if *target*  
 81816 exists and does not name a directory.

81817 In the following description, the term *dest\_file* refers to the file named by the destination path.  
 81818 The term *source\_file* refers to the file that is being copied, whether specified as an operand or a  
 81819 file in a file hierarchy rooted in a *source\_file* operand. If *source\_file* is a file of type symbolic link:

- 81820 • If the **-R** option was not specified, *cp* shall take actions based on the type and contents of  
 81821 the file referenced by the symbolic link, and not by the symbolic link itself, unless the **-P**  
 81822 option was specified.
- 81823 • If the **-R** option was specified:
  - 81824 — If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**,  
 81825 **-L**, or **-P** will be used as a default.
  - 81826 — If the **-H** option was specified, *cp* shall take actions based on the type and contents of  
 81827 the file referenced by any symbolic link specified as a *source\_file* operand.
  - 81828 — If the **-L** option was specified, *cp* shall take actions based on the type and contents of  
 81829 the file referenced by any symbolic link specified as a *source\_file* operand or any  
 81830 symbolic links encountered during traversal of a file hierarchy.

81831 — If the `-P` option was specified, `cp` shall copy any symbolic link specified as a  
 81832 `source_file` operand and any symbolic links encountered during traversal of a file  
 81833 hierarchy, and shall not follow any symbolic links.

81834 For each `source_file`, the following steps shall be taken:

- 81835 1. If `source_file` references the same file as `dest_file`, `cp` may write a diagnostic message to  
 81836 standard error; it shall do nothing more with `source_file` and shall go on to any remaining  
 81837 files.
- 81838 2. If `source_file` is of type directory, the following steps shall be taken:
  - 81839 a. If the `-R` option was not specified, `cp` shall write a diagnostic message to standard  
 81840 error, do nothing more with `source_file`, and go on to any remaining files.
  - 81841 b. If `source_file` was not specified as an operand and `source_file` is dot or dot-dot, `cp`  
 81842 shall do nothing more with `source_file` and go on to any remaining files.
  - 81843 c. If `dest_file` exists and it is a file type not specified by the System Interfaces volume  
 81844 of POSIX.1-2008, the behavior is implementation-defined.
  - 81845 d. If `dest_file` exists and it is not of type directory, `cp` shall write a diagnostic message  
 81846 to standard error, do nothing more with `source_file` or any files below `source_file` in  
 81847 the file hierarchy, and go on to any remaining files.
  - 81848 e. If the directory `dest_file` does not exist, it shall be created with file permission bits  
 81849 set to the same value as those of `source_file`, modified by the file creation mask of  
 81850 the user if the `-p` option was not specified, and then bitwise-inclusively OR'ed  
 81851 with `S_IRWXU`. If `dest_file` cannot be created, `cp` shall write a diagnostic message to  
 81852 standard error, do nothing more with `source_file`, and go on to any remaining files.  
 81853 It is unspecified if `cp` attempts to copy files in the file hierarchy rooted in `source_file`.
  - 81854 f. The files in the directory `source_file` shall be copied to the directory `dest_file`, taking  
 81855 the four steps (1 to 4) listed here with the files as `source_files`.
  - 81856 g. If `dest_file` was created, its file permission bits shall be changed (if necessary) to be  
 81857 the same as those of `source_file`, modified by the file creation mask of the user if the  
 81858 `-p` option was not specified.
  - 81859 h. The `cp` utility shall do nothing more with `source_file` and go on to any remaining  
 81860 files.
- 81861 3. If `source_file` is of type regular file, the following steps shall be taken:
  - 81862 a. The behavior is unspecified if `dest_file` exists and was written by a previous step.  
 81863 Otherwise, if `dest_file` exists, the following steps shall be taken:
    - 81864 i. If the `-i` option is in effect, the `cp` utility shall write a prompt to the standard  
 81865 error and read a line from the standard input. If the response is not  
 81866 affirmative, `cp` shall do nothing more with `source_file` and go on to any  
 81867 remaining files.
    - 81868 ii. A file descriptor for `dest_file` shall be obtained by performing actions  
 81869 equivalent to the `open()` function defined in the System Interfaces volume of  
 81870 POSIX.1-2008 called using `dest_file` as the `path` argument, and the bitwise-  
 81871 inclusive OR of `O_WRONLY` and `O_TRUNC` as the `oflag` argument.
    - 81872 iii. If the attempt to obtain a file descriptor fails and the `-f` option is in effect, `cp`  
 81873 shall attempt to remove the file by performing actions equivalent to the  
 81874 `unlink()` function defined in the System Interfaces volume of POSIX.1-2008

- 81875 called using *dest\_file* as the *path* argument. If this attempt succeeds, *cp* shall  
81876 continue with step 3b.
- 81877 b. If *dest\_file* does not exist, a file descriptor shall be obtained by performing actions  
81878 equivalent to the *open()* function defined in the System Interfaces volume of  
81879 POSIX.1-2008 called using *dest\_file* as the *path* argument, and the bitwise-inclusive  
81880 OR of *O\_WRONLY* and *O\_CREAT* as the *oflag* argument. The file permission bits  
81881 of *source\_file* shall be the *mode* argument.
- 81882 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to  
81883 standard error, do nothing more with *source\_file*, and go on to any remaining files.
- 81884 d. The contents of *source\_file* shall be written to the file descriptor. Any write errors  
81885 shall cause *cp* to write a diagnostic message to standard error and continue to step  
81886 3e.
- 81887 e. The file descriptor shall be closed.
- 81888 f. The *cp* utility shall do nothing more with *source\_file*. If a write error occurred in  
81889 step 3d, it is unspecified if *cp* continues with any remaining files. If no write error  
81890 occurred in step 3d, *cp* shall go on to any remaining files.
- 81891 4. Otherwise, the **-R** option was specified, and the following steps shall be taken:
- 81892 a. The *dest\_file* shall be created with the same file type as *source\_file*.
- 81893 b. If *source\_file* is a file of type FIFO, the file permission bits shall be the same as those  
81894 of *source\_file*, modified by the file creation mask of the user if the **-p** option was not  
81895 specified. Otherwise, the permissions, owner ID, and group ID of *dest\_file* are  
81896 implementation-defined.
- 81897 If this creation fails for any reason, *cp* shall write a diagnostic message to standard  
81898 error, do nothing more with *source\_file*, and go on to any remaining files.
- 81899 c. If *source\_file* is a file of type symbolic link, and the options require the symbolic link  
81900 itself to be acted upon, the pathname contained in *dest\_file* shall be the same as the  
81901 pathname contained in *source\_file*.
- 81902 If this fails for any reason, *cp* shall write a diagnostic message to standard error, do  
81903 nothing more with *source\_file*, and go on to any remaining files.
- 81904 If the implementation provides additional or alternate access control mechanisms (see XBD  
81905 [Section 4.4](#), on page 108), their effect on copies of files is implementation-defined.

**OPTIONS**

- 81906 The *cp* utility shall conform to XBD [Section 12.2](#) (on page 215).
- 81907 The following options shall be supported:
- 81908
- 81909 **-f** If a file descriptor for a destination file cannot be obtained, as described in step  
81910 3.a.ii., attempt to unlink the destination file and proceed.
- 81911 **-H** Take actions based on the type and contents of the file referenced by any symbolic  
81912 link specified as a *source\_file* operand.
- 81913 **-i** Write a prompt to standard error before copying to any existing non-directory  
81914 destination file. If the response from the standard input is affirmative, the copy  
81915 shall be attempted; otherwise, it shall not.

|       |                              |                                                                                                                                                                                                                                      |
|-------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 81916 | -L                           | Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand or any symbolic links encountered during traversal of a file hierarchy.                            |
| 81917 |                              |                                                                                                                                                                                                                                      |
| 81918 |                              |                                                                                                                                                                                                                                      |
| 81919 | -P                           | Take actions on any symbolic link specified as a <i>source_file</i> operand or any symbolic link encountered during traversal of a file hierarchy.                                                                                   |
| 81920 |                              |                                                                                                                                                                                                                                      |
| 81921 | -p                           | Duplicate the following characteristics of each source file in the corresponding destination file:                                                                                                                                   |
| 81922 |                              |                                                                                                                                                                                                                                      |
| 81923 |                              | 1. The time of last data modification and time of last access. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.                                                               |
| 81924 |                              |                                                                                                                                                                                                                                      |
| 81925 |                              | 2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.                                                                            |
| 81926 |                              |                                                                                                                                                                                                                                      |
| 81927 |                              | 3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error. |
| 81928 |                              |                                                                                                                                                                                                                                      |
| 81929 |                              |                                                                                                                                                                                                                                      |
| 81930 |                              | If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but                                                                 |
| 81931 |                              | are not duplicated in the destination file, it is unspecified whether <i>cp</i> writes a                                                                                                                                             |
| 81932 |                              | diagnostic message to standard error.                                                                                                                                                                                                |
| 81933 |                              |                                                                                                                                                                                                                                      |
| 81934 |                              | The order in which the preceding characteristics are duplicated is unspecified. The                                                                                                                                                  |
| 81935 |                              | <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved.                                                                                                                                                  |
| 81936 | -R                           | Copy file hierarchies.                                                                                                                                                                                                               |
| 81937 |                              | Specifying more than one of the mutually-exclusive options -H, -L, and -P shall not be                                                                                                                                               |
| 81938 |                              | considered an error. The last option specified shall determine the behavior of the utility.                                                                                                                                          |
| 81939 | <b>OPERANDS</b>              |                                                                                                                                                                                                                                      |
| 81940 |                              | The following operands shall be supported:                                                                                                                                                                                           |
| 81941 | <i>source_file</i>           | A pathname of a file to be copied. If a <i>source_file</i> operand is '-', it shall refer to a                                                                                                                                       |
| 81942 |                              | file named -; implementations shall not treat it as meaning standard input.                                                                                                                                                          |
| 81943 | <i>target_file</i>           | A pathname of an existing or nonexistent file, used for the output when a single                                                                                                                                                     |
| 81944 |                              | file is copied. If a <i>target_file</i> operand is '-', it shall refer to a file named -;                                                                                                                                            |
| 81945 |                              | implementations shall not treat it as meaning standard output.                                                                                                                                                                       |
| 81946 | <i>target</i>                | A pathname of a directory to contain the copied files.                                                                                                                                                                               |
| 81947 | <b>STDIN</b>                 |                                                                                                                                                                                                                                      |
| 81948 |                              | The standard input shall be used to read an input line in response to each prompt specified in                                                                                                                                       |
| 81949 |                              | the STDERR section. Otherwise, the standard input shall not be used.                                                                                                                                                                 |
| 81950 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                      |
| 81951 |                              | The input files specified as operands may be of any file type.                                                                                                                                                                       |
| 81952 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                      |
| 81953 |                              | The following environment variables shall affect the execution of <i>cp</i> :                                                                                                                                                        |
| 81954 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null.                                                                                                                                               |
| 81955 |                              | (See XBD Section 8.2 (on page 174) for the precedence of internationalization                                                                                                                                                        |
| 81956 |                              | variables used to determine the values of locale categories.)                                                                                                                                                                        |

|       |                               |                                                                                                                                                                                                                                                                                                                                                          |
|-------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 81957 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                 |
| 81958 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81959 | <i>LC_COLLATE</i>             |                                                                                                                                                                                                                                                                                                                                                          |
| 81960 |                               | Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <i>LC_MESSAGES</i> category.                                                                                                                   |
| 81961 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81962 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81963 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <i>LC_MESSAGES</i> category. |
| 81964 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81965 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81966 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81967 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81968 | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                                                                          |
| 81969 |                               | Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.                                                                                                                                                                          |
| 81970 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81971 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81972 | XSI <i>NLSPATH</i>            | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                    |
| 81973 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                          |
| 81974 |                               | Default.                                                                                                                                                                                                                                                                                                                                                 |
| 81975 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                          |
| 81976 |                               | Not used.                                                                                                                                                                                                                                                                                                                                                |
| 81977 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                          |
| 81978 |                               | A prompt shall be written to standard error under the conditions specified in the DESCRIPTION section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.                                                                             |
| 81979 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81980 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81981 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                          |
| 81982 |                               | The output files may be of any type.                                                                                                                                                                                                                                                                                                                     |
| 81983 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                          |
| 81984 |                               | None.                                                                                                                                                                                                                                                                                                                                                    |
| 81985 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                          |
| 81986 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                             |
| 81987 |                               | 0 All files were copied successfully.                                                                                                                                                                                                                                                                                                                    |
| 81988 |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                    |
| 81989 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                          |
| 81990 |                               | If <b>cp</b> is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.                                                                                                                                           |
| 81991 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 81992 |                               |                                                                                                                                                                                                                                                                                                                                                          |

81993 **APPLICATION USAGE**

81994 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to  
 81995 prevent users from creating programs that are set-user-ID or set-group-ID to them when copying  
 81996 files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example,  
 81997 if a file is set-user-ID and the copy has a different group ID than the source, a new group of users  
 81998 has execute permission to a set-user-ID program than did previously. In particular, this is a  
 81999 problem for superusers copying users' trees.

82000 **EXAMPLES**

82001 None.

82002 **RATIONALE**

82003 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally  
 82004 removing files when copying. Although the 4.3 BSD version does not prompt if the standard  
 82005 input is not a terminal, the standard developers decided that use of `-i` is a request for  
 82006 interaction, so when the destination path exists, the utility takes instructions from whatever  
 82007 responds on standard input.

82008 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 82009 contents of prompts are specified because implementations may desire more descriptive  
 82010 prompts than those used on historical implementations. Therefore, an application using the `-i`  
 82011 option relies on the system to provide the most suitable dialog directly with the user, based on  
 82012 the behavior specified.

82013 The `-p` option is historical practice on BSD systems, duplicating the time of last data  
 82014 modification and time of last access. This volume of POSIX.1-2008 extends it to preserve the user  
 82015 and group IDs, as well as the file permissions. This requirement has obvious problems in that  
 82016 the directories are almost certainly modified after being copied. This volume of POSIX.1-2008  
 82017 requires that the modification times be preserved. The statement that the order in which the  
 82018 characteristics are duplicated is unspecified is to permit implementations to provide the  
 82019 maximum amount of security for the user. Implementations should take into account the  
 82020 obvious security issues involved in setting the owner, group, and mode in the wrong order or  
 82021 creating files with an owner, group, or mode different from the final value.

82022 It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be  
 82023 set due to the widespread practice of users using `-p` to duplicate some portion of the file  
 82024 characteristics, indifferent to the duplication of others. Historic implementations only write  
 82025 diagnostic messages on errors other than [EPERM].

82026 Earlier versions of this standard included support for the `-r` option to copy file hierarchies. The  
 82027 `-r` option is historical practice on BSD and BSD-derived systems. This option is no longer  
 82028 specified by POSIX.1-2008 but may be present in some implementations. The `-R` option was  
 82029 added as a close synonym to the `-r` option, selected for consistency with all other options in this  
 82030 volume of POSIX.1-2008 that do recursive directory descent.

82031 The difference between `-R` and the removed `-r` option is in the treatment by `cp` of file types other  
 82032 than regular and directory. It was implementation-defined how the `-` option treated special files  
 82033 to allow both historical implementations and those that chose to support `-r` with the same  
 82034 abilities as `-R` defined by this volume of POSIX.1-2008. The original `-r` flag, for historic reasons,  
 82035 did not handle special files any differently from regular files, but always read the file and copied  
 82036 its contents. This had obvious problems in the presence of special file types; for example,  
 82037 character devices, FIFOs, and sockets.

82038 When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy  
 82039 files that are on the same level in the hierarchy or above the file where the failure occurred. It is  
 82040 unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which

82041 cannot succeed in any case).

82042 Permissions, owners, and groups of created special file types have been deliberately left as  
82043 implementation-defined. This is to allow systems to satisfy special requirements (for example,  
82044 allowing users to create character special devices, but requiring them to be owned by a certain  
82045 group). In general, it is strongly suggested that the permissions, owner, and group be the same  
82046 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable  
82047 that additional privileges are required to create block, character, or other implementation-  
82048 defined special file types.

82049 Additionally, the *-p* option explicitly requires that all set-user-ID and set-group-ID permissions  
82050 be discarded if any of the owner or group IDs cannot be set. This is to keep users from  
82051 unintentionally giving away special privilege when copying programs.

82052 When creating regular files, historical versions of *cp* use the mode of the source file as modified  
82053 by the file mode creation mask. Other choices would have been to use the mode of the source file  
82054 unmodified by the creation mask or to use the same mode as would be given to a new file  
82055 created by the user (plus the execution bits of the source file) and then modify it by the file mode  
82056 creation mask. In the absence of any strong reason to change historic practice, it was in large part  
82057 retained.

82058 When creating directories, historical versions of *cp* use the mode of the source directory, plus  
82059 read, write, and search bits for the owner, as modified by the file mode creation mask. This is  
82060 done so that *cp* can copy trees where the user has read permission, but the owner does not. A  
82061 side-effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the  
82062 copy is done, historical versions of *cp* set the permissions on the created directory to be the same  
82063 as the source directory, unmodified by the file creation mask.

82064 This behavior has been modified so that *cp* is always able to create the contents of the directory,  
82065 regardless of the file creation mask. After the copy is done, the permissions are set to be the same  
82066 as the source directory, as modified by the file creation mask. This latter change from historical  
82067 behavior is to prevent users from accidentally creating directories with permissions beyond  
82068 those they would normally set and for consistency with the behavior of *cp* in creating files.

82069 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations  
82070 are strongly encouraged to do so. Historical implementations have detected the attempt in most  
82071 cases.

82072 There are two methods of copying subtrees in this volume of POSIX.1-2008. The other method is  
82073 described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility  
82074 provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each  
82075 provides additional functionality to the other; in particular, *pax* maintains the hard-link structure  
82076 of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results  
82077 be similar (using appropriate option combinations in both utilities). The results are not required  
82078 to be identical; there seemed insufficient gain to applications to balance the difficulty of  
82079 implementations having to guarantee that the results would be exactly identical.

82080 The wording allowing *cp* to copy a directory to implementation-defined file types not specified  
82081 by the System Interfaces volume of POSIX.1-2008 is provided so that implementations  
82082 supporting symbolic links are not required to prohibit copying directories to symbolic links.  
82083 Other extensions to the System Interfaces volume of POSIX.1-2008 file types may need to use  
82084 this loophole as well.

82085 **FUTURE DIRECTIONS**

82086 None.

82087 **SEE ALSO**82088 *mv, find, ln, pax*82089 XBD [Section 4.4](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)82090 XSH *open()*, *unlink()*82091 **CHANGE HISTORY**

82092 First released in Issue 2.

82093 **Issue 6**82094 The `-r` option is marked obsolescent.82095 The new options `-H`, `-L`, and `-P` are added to align with the IEEE P1003.2b draft standard. These  
82096 options affect the processing of symbolic links.82097 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the `-P` option.82098 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the  
82099 SEE ALSO section.82100 **Issue 7**82101 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
82102 `LC_MESSAGES` environment variable.

82103 Austin Group Interpretations 1003.1-2001 #092, #164, #165, and #168 are applied.

82104 SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

82105 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82106 SD5-XCU-ERN-102 is applied, clarifying the `-i` option within the OPTIONS section.82107 The obsolescent `-r` option is removed.82108 The `-P` option is added to the SYNOPSIS and to the DESCRIPTION with respect to the `-R`  
82109 option.

82110 **NAME**

82111 crontab — schedule periodic background work

82112 **SYNOPSIS**82113 crontab [*file*]82114 UP crontab [**-e**|-l|-r]82115 **DESCRIPTION**

82116 UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of  
 82117 commands and the times at which they shall be executed. The new crontab entry can be input by  
 82118 UP specifying *file* or input from standard input if no *file* operand is specified, or by using an editor,  
 82119 if **-e** is specified.

82120 Upon execution of a command from a crontab entry, the implementation shall supply a default  
 82121 environment, defining at least the following environment variables:

82122 *HOME* A pathname of the user's home directory.82123 *LOGNAME* The user's login name.82124 *PATH* A string representing a search path guaranteed to find all of the standard utilities.82125 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by  
 82126 this volume of POSIX.1-2008, the value shall be a pathname for *sh*.

82127 The values of these variables when *crontab* is invoked as specified by this volume of  
 82128 POSIX.1-2008 shall not affect the default values provided when the scheduled command is run.

82129 If standard output and standard error are not redirected by commands executed from the  
 82130 crontab entry, any generated output or errors shall be mailed, via an implementation-defined  
 82131 method, to the user.

82132 XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is  
 82133 located in an implementation-defined directory. If that file does not exist, the file **cron.deny**,  
 82134 which is located in an implementation-defined directory, shall be checked to determine whether  
 82135 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate  
 82136 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage  
 82137 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

82138 **OPTIONS**82139 The *crontab* utility shall conform to XBD Section 12.2 (on page 215).

82140 The following options shall be supported:

82141 UP **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if  
 82142 the crontab entry does not exist. When editing is complete, the entry shall be  
 82143 installed as the user's crontab entry.

82144 **-l** (The letter ell.) List the invoking user's crontab entry.82145 **-r** Remove the invoking user's crontab entry.82146 **OPERANDS**

82147 The following operand shall be supported:

82148 *file* The pathname of a file that contains specifications, in the format defined in the  
 82149 INPUT FILES section, for crontab entries.

**crontab**82150 **STDIN**

82151 See the INPUT FILES section.

82152 **INPUT FILES**82153 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file  
82154 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The  
82155 first five fields shall be integer patterns that specify the following:

- 82156 1. Minute [0,59]
- 82157 2. Hour [0,23]
- 82158 3. Day of the month [1,31]
- 82159 4. Month of the year [1,12]
- 82160 5. Day of the week ([0,6] with 0=Sunday)

82161 Each of these patterns can be either an <asterisk> (meaning all valid values), an element, or a list  
82162 of elements separated by <comma> characters. An element shall be either a number or two  
82163 numbers separated by a <hyphen> (meaning an inclusive range). The specification of days can  
82164 be made by two fields (day of the month and day of the week). If month, day of month, and day  
82165 of week are all <asterisk> characters, every day shall be matched. If either the month or day of  
82166 month is specified as an element or list, but the day of week is an <asterisk>, the month and day  
82167 of month fields shall specify the days that match. If both month and day of month are specified  
82168 as an <asterisk>, but day of week is an element or list, then only the specified days of the week  
82169 match. Finally, if either the month or day of month is specified as an element or list, and the day  
82170 of week is also specified as an element or list, then any day matching either the month and day  
82171 of month, or the day of week, shall be matched.

82172 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified  
82173 times. A <percent-sign> character in this field shall be translated to a <newline>. Any character  
82174 preceded by a <backslash> (including the '%') shall cause that character to be treated literally.  
82175 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the  
82176 command interpreter. The other lines shall be made available to the command as standard input.

82177 Blank lines and those whose first non-&lt;blank&gt; is '#' shall be ignored.

82178 XSI The text files **crontab.allow** and **crontab.deny**, which are located in an implementation-defined  
82179 directory, shall contain zero or more user names, one per line, of users who are, respectively,  
82180 authorized or denied access to the service underlying the *crontab* utility.

82181 **ENVIRONMENT VARIABLES**82182 The following environment variables shall affect the execution of *crontab*:

- |       |                 |                                                                                                                                                                                                                                    |
|-------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 82183 | <i>EDITOR</i>   | Determine the editor to be invoked when the <b>-e</b> option is specified. The default editor shall be <i>vi</i> .                                                                                                                 |
| 82184 |                 |                                                                                                                                                                                                                                    |
| 82185 | <i>LANG</i>     | Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.) |
| 82186 |                 |                                                                                                                                                                                                                                    |
| 82187 |                 |                                                                                                                                                                                                                                    |
| 82188 | <i>LC_ALL</i>   | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                           |
| 82189 |                 |                                                                                                                                                                                                                                    |
| 82190 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                          |
| 82191 |                 |                                                                                                                                                                                                                                    |
| 82192 |                 |                                                                                                                                                                                                                                    |

|       |     |                               |                                                                                                                           |
|-------|-----|-------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| 82193 |     | <i>LC_MESSAGES</i>            |                                                                                                                           |
| 82194 |     |                               | Determine the locale that should be used to affect the format and contents of                                             |
| 82195 |     |                               | diagnostic messages written to standard error.                                                                            |
| 82196 | XSI | <i>NLSPATH</i>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                     |
| 82197 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                           |
| 82198 |     |                               | Default.                                                                                                                  |
| 82199 |     | <b>STDOUT</b>                 |                                                                                                                           |
| 82200 |     |                               | If the <code>-l</code> option is specified, the crontab entry shall be written to the standard output.                    |
| 82201 |     | <b>STDERR</b>                 |                                                                                                                           |
| 82202 |     |                               | The standard error shall be used only for diagnostic messages.                                                            |
| 82203 |     | <b>OUTPUT FILES</b>           |                                                                                                                           |
| 82204 |     |                               | None.                                                                                                                     |
| 82205 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                           |
| 82206 |     |                               | None.                                                                                                                     |
| 82207 |     | <b>EXIT STATUS</b>            |                                                                                                                           |
| 82208 |     |                               | The following exit values shall be returned:                                                                              |
| 82209 |     | 0                             | Successful completion.                                                                                                    |
| 82210 |     | >0                            | An error occurred.                                                                                                        |
| 82211 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                           |
| 82212 | UP  |                               | The user's crontab entry is not submitted, removed, <b>edited</b> , or listed.                                            |
| 82213 |     | <b>APPLICATION USAGE</b>      |                                                                                                                           |
| 82214 |     |                               | The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other                                 |
| 82215 |     |                               | cultures may be supported with substantially different interfaces, although implementations are                           |
| 82216 |     |                               | encouraged to provide comparable levels of functionality.                                                                 |
| 82217 |     |                               | The default settings of the <i>HOME</i> , <i>LOGNAME</i> , <i>PATH</i> , and <i>SHELL</i> variables that are given to the |
| 82218 |     |                               | scheduled job are not affected by the settings of those variables when <i>crontab</i> is run; as stated,                  |
| 82219 |     |                               | they are defaults. The text about "invoked as specified by this volume of POSIX.1-2008" means                             |
| 82220 |     |                               | that the implementation may provide extensions that allow these variables to be affected at                               |
| 82221 |     |                               | runtime, but that the user has to take explicit action in order to access the extension, such as give                     |
| 82222 |     |                               | a new option flag or modify the format of the crontab entry.                                                              |
| 82223 |     |                               | A typical user error is to type only <i>crontab</i> ; this causes the system to wait for the new crontab                  |
| 82224 |     |                               | entry on standard input. If end-of-file is typed (generally <code>&lt;control&gt;-D</code> ), the crontab entry is        |
| 82225 |     |                               | replaced by an empty file. In this case, the user should type the interrupt character, which                              |
| 82226 |     |                               | prevents the crontab entry from being replaced.                                                                           |
| 82227 |     | <b>EXAMPLES</b>               |                                                                                                                           |
| 82228 |     | 1.                            | Clean up <b>core</b> files every weekday morning at 3:15 am:                                                              |
| 82229 |     |                               | <code>15 3 * * 1-5 find "\$HOME" -name core -exec rm -f {} + 2&gt;/dev/null</code>                                        |
| 82230 |     | 2.                            | Mail a birthday greeting:                                                                                                 |
| 82231 |     |                               | <code>0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.</code>                                                       |
| 82232 |     | 3.                            | As an example of specifying the two types of days:                                                                        |
| 82233 |     |                               | <code>0 0 1,15 * 1</code>                                                                                                 |

82234 would run a command on the first and fifteenth of each month, as well as on every  
 82235 Monday. To specify days by only one field, the other field should be set to '\*'; for  
 82236 example:

82237 0 0 \* \* 1

82238 would run a command only on Mondays.

#### 82239 RATIONALE

82240 All references to a *cron* daemon and to *cron files* have been omitted. Although historical  
 82241 implementations have used this arrangement, there is no reason to limit future implementations.

82242 This description of *crontab* is designed to support only users with normal privileges. The format  
 82243 of the input is based on the System V *crontab*; however, there is no requirement here that the  
 82244 actual system database used by the *cron* daemon (or a similar mechanism) use this format  
 82245 internally. For example, systems derived from BSD are likely to have an additional field  
 82246 appended that indicates the user identity to be used when the job is submitted.

82247 The *-e* option was adopted from the SVID as a user convenience, although it does not exist in all  
 82248 historical implementations.

#### 82249 FUTURE DIRECTIONS

82250 None.

#### 82251 SEE ALSO

82252 *at*

82253 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

#### 82254 CHANGE HISTORY

82255 First released in Issue 2.

#### 82256 Issue 6

82257 This utility is marked as part of the User Portability Utilities option.

82258 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 82259 Issue 7

82260 The *crontab* utility (except for the *-e* option) is moved from the User Portability Utilities option  
 82261 to the Base. User Portability Utilities is now an option for interactive utilities.

82262 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
 82263 by the *crontab* utility.

82264 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82265 The first example is changed to remove the unreliable use of *find | xargs*.

## 82266 NAME

82267 csplit — split files based on context

## 82268 SYNOPSIS

82269 csplit [-ks] [-f *prefix*] [-n *number*] *file arg...*

## 82270 DESCRIPTION

82271 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into  
82272 other files as directed by the *arg* operands, and write the sizes of the files.

## 82273 OPTIONS

82274 The *csplit* utility shall conform to XBD Section 12.2 (on page 215).

82275 The following options shall be supported:

82276 **-f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is *xx00* ... *xxn*. If  
82277 the *prefix* argument would create a filename exceeding {NAME\_MAX} bytes, an  
82278 error shall result, *csplit* shall exit with a diagnostic message and no files shall be  
82279 created.82280 **-k** Leave previously created files intact. By default, *csplit* shall remove created files if  
82281 an error occurs.82282 **-n *number*** Use *number* decimal digits to form filenames for the file pieces. The default shall be  
82283 2.82284 **-s** Suppress the output of file size messages.

## 82285 OPERANDS

82286 The following operands shall be supported:

82287 *file* The pathname of a text file to be split. If *file* is '-', the standard input shall be  
82288 used.82289 Each *arg* operand can be one of the following:82290 */rexp/[offset]*82291 A file shall be created using the content of the lines from the current line up to, but  
82292 not including, the line that results from the evaluation of the regular expression  
82293 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for  
82294 basic regular expressions described in XBD Section 9.3 (on page 183). The  
82295 application shall use the sequence "\/" to specify a <slash> character within the  
82296 *rexp*. The optional offset shall be a positive or negative integer value representing a  
82297 number of lines. A positive integer value can be preceded by '+'. If the selection  
82298 of lines from an *offset* expression of this type would create a file with zero lines, or  
82299 one with greater than the number of lines left in the input file, the results are  
82300 unspecified. After the section is created, the current line shall be set to the line that  
82301 results from the evaluation of the regular expression with any offset applied. If the  
82302 current line is the first line in the file and a regular expression operation has not yet  
82303 been performed, the pattern match of *rexp* shall be applied from the current line to  
82304 the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the  
82305 line following the current line to the end of the file.82306 *%rexp%[offset]*82307 Equivalent to */rexp/[offset]*, except that no file shall be created for the selected  
82308 section of the input file. The application shall use the sequence "\%" to specify a  
82309 <percent-sign> character within the *rexp*.

**csplit**

Utilities

- 82310 *line\_no* Create a file from the current line up to (but not including) the line number *line\_no*.  
 82311 Lines in the file shall be numbered starting at one. The current line becomes  
 82312 *line\_no*.
- 82313 {*num*} Repeat operand. This operand can follow any of the operands described  
 82314 previously. If it follows a *rexp* type operand, that operand shall be applied *num*  
 82315 more times. If it follows a *line\_no* operand, the file shall be split every *line\_no* lines,  
 82316 *num* times, from that point.
- 82317 An error shall be reported if an operand does not reference a line between the current position  
 82318 and the end of the file.
- 82319 **STDIN**
- 82320 See the INPUT FILES section.
- 82321 **INPUT FILES**
- 82322 The input file shall be a text file.
- 82323 **ENVIRONMENT VARIABLES**
- 82324 The following environment variables shall affect the execution of *csplit*:
- 82325 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 82326 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 82327 variables used to determine the values of locale categories.)
- 82328 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 82329 internationalization variables.
- 82330 *LC\_COLLATE*
- 82331 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 82332 character collating elements within regular expressions.
- 82333 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 82334 characters (for example, single-byte as opposed to multi-byte characters in  
 82335 arguments and input files) and the behavior of character classes within regular  
 82336 expressions.
- 82337 *LC\_MESSAGES*
- 82338 Determine the locale that should be used to affect the format and contents of  
 82339 diagnostic messages written to standard error.
- 82340 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 82341 **ASYNCHRONOUS EVENTS**
- 82342 If the *-k* option is specified, created files shall be retained. Otherwise, the default action occurs.
- 82343 **STDOUT**
- 82344 Unless the *-s* option is used, the standard output shall consist of one line per file created, with a  
 82345 format as follows:
- 82346 "%d\n", <file size in bytes>
- 82347 **STDERR**
- 82348 The standard error shall be used only for diagnostic messages.
- 82349 **OUTPUT FILES**
- 82350 The output files shall contain portions of the original input file; otherwise, unchanged.

82351 **EXTENDED DESCRIPTION**

82352 None.

82353 **EXIT STATUS**

82354 The following exit values shall be returned:

82355 0 Successful completion.

82356 &gt;0 An error occurred.

82357 **CONSEQUENCES OF ERRORS**82358 By default, created files shall be removed if an error occurs. When the `-k` option is specified,  
82359 created files shall not be removed if an error occurs.82360 **APPLICATION USAGE**

82361 None.

82362 **EXAMPLES**82363 1. This example creates four files, **cobol00** ... **cobol03**:82364 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

82365 After editing the split files, they can be recombined as follows:

82366 `cat cobol0[0-3] > file`

82367 Note that this example overwrites the original file.

82368 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up  
82369 to 9 999 lines; this is because lines in the file are numbered from 1 rather than zero, for  
82370 historical reasons:82371 `csplit -k file 100 {99}`82372 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with  
82373 a `'}'` at the beginning of the line, this example creates a file containing each separate C  
82374 routine (up to 21) in **prog.c**:82375 `csplit -k prog.c '%main(%' '/^}/+1' {20}`82376 **RATIONALE**82377 The `-n` option was added to extend the range of filenames that could be handled.82378 Consideration was given to adding a `-a` flag to use the alphabetic filename generation used by  
82379 the historical *split* utility, but the functionality added by the `-n` option was deemed to make  
82380 alphabetic naming unnecessary.82381 **FUTURE DIRECTIONS**

82382 None.

82383 **SEE ALSO**82384 *sed*, *split*

82385 XBD Chapter 8 (on page 173), Section 9.3 (on page 183), Section 12.2 (on page 215)

82386 **CHANGE HISTORY**

82387 First released in Issue 2.

**csplit**

Utilities

- 82388 **Issue 5**  
82389 The FUTURE DIRECTIONS section is added.
- 82390 **Issue 6**  
82391 This utility is marked as part of the User Portability Utilities option.  
82392 The APPLICATION USAGE section is added.  
82393 The description of regular expression operands is changed to align with the IEEE P1003.2b draft  
82394 standard.  
82395 The normative text is reworded to avoid use of the term “must” for application requirements.
- 82396 **Issue 7**  
82397 The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability  
82398 Utilities is now an option for interactive utilities.  
82399 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.  
82400 The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two  
82401 pieces.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

82402 **NAME**

82403 ctags — create a tags file (DEVELOPMENT, FORTRAN)

82404 **SYNOPSIS**82405 SD `ctags [-a] [-f tagsfile] pathname...`82406 `ctags -x pathname...`82407 **DESCRIPTION**

82408 The *ctags* utility shall be provided on systems that support the the Software Development  
 82409 Utilities option, and either or both of the C-Language Development Utilities option and  
 82410 FORTRAN Development Utilities option. On other systems, it is optional.

82411 The *ctags* utility shall write a *tagsfile* or an index of objects from C-language or FORTRAN source  
 82412 files specified by the *pathname* operands. The *tagsfile* shall list the locators of language-specific  
 82413 objects within the source files. A locator consists of a name, *pathname*, and either a search  
 82414 pattern or a line number that can be used in searching for the object definition. The objects that  
 82415 shall be recognized are specified in the EXTENDED DESCRIPTION section.

82416 **OPTIONS**82417 The *ctags* utility shall conform to XBD Section 12.2 (on page 215).

82418 The following options shall be supported:

82419 **-a** Append to *tagsfile*.82420 **-f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in the  
82421 current directory.82422 **-x** Produce a list of object names, the line number, and filename in which each is  
82423 defined, as well as the text of that line, and write this to the standard output. A  
82424 *tagsfile* shall not be created when **-x** is specified.82425 **OPERANDS**82426 The following *pathname* operands are supported:82427 ***file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language  
82428 source code. Such files that are not valid input to *c99* produce unspecified results.82429 ***file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language  
82430 source code. Such files that are not valid input to *c99* produce unspecified results.82431 ***file.f*** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-  
82432 language source code. Such files that are not valid input to *fort77* produce  
82433 unspecified results.

82434 The handling of other files is implementation-defined.

82435 **STDIN**

82436 See the INPUT FILES section.

82437 **INPUT FILES**82438 The input files shall be text files containing source code in the language indicated by the  
82439 operand filename suffixes.82440 **ENVIRONMENT VARIABLES**

- 82441 The following environment variables shall affect the execution of *ctags*:
- 82442 *LANG* Provide a default value for the internationalization variables that are unset or null.  
82443 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
82444 variables used to determine the values of locale categories.)
- 82445 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
82446 internationalization variables.
- 82447 *LC\_COLLATE*  
82448 Determine the order in which output is sorted for the *-x* option. The POSIX locale  
82449 determines the order in which the *tagsfile* is written.
- 82450 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
82451 characters (for example, single-byte as opposed to multi-byte characters in  
82452 arguments and input files). When processing C-language source code, if the locale  
82453 is not compatible with the C locale described by the ISO C standard, the results are  
82454 unspecified.
- 82455 *LC\_MESSAGES*  
82456 Determine the locale that should be used to affect the format and contents of  
82457 diagnostic messages written to standard error.
- 82458 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 82459 **ASYNCHRONOUS EVENTS**
- 82460 Default.
- 82461 **STDOUT**
- 82462 The list of object name information produced by the *-x* option shall be written to standard  
82463 output in the following format:
- 82464 "%s %d %s %s", <object-name>, <line-number>, <filename>, <text>
- 82465 where <text> is the text of line <line-number> of file <filename>.
- 82466 **STDERR**
- 82467 The standard error shall be used only for diagnostic messages.
- 82468 **OUTPUT FILES**
- 82469 When the *-x* option is not specified, the format of the output file shall be:
- 82470 "%s\t%s\t/ %s/\n", <identifier>, <filename>, <pattern>
- 82471 where <pattern> is a search pattern that could be used by an editor to find the defining instance  
82472 of <identifier> in <filename> (where *defining instance* is indicated by the declarations listed in the  
82473 EXTENDED DESCRIPTION).
- 82474 An optional <circumflex> ('*^*') can be added as a prefix to <pattern>, and an optional <dollar-  
82475 sign> can be appended to <pattern> to indicate that the pattern is anchored to the beginning  
82476 (end) of a line of text. Any <slash> or <backslash> characters in <pattern> shall be preceded by a  
82477 <backslash> character. The anchoring <circumflex>, <dollar-sign>, and escaping <backslash>  
82478 characters shall not be considered part of the search pattern. All other characters in the search  
82479 pattern shall be considered literal characters.

82480 An alternative format is:

82481 "%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>

82482 which is identical to the first format except that <slash> characters in <pattern> shall not be  
82483 preceded by escaping <backslash> characters, and <question-mark> characters in <pattern>  
82484 shall be preceded by <backslash> characters.

82485 A second alternative format is:

82486 "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>

82487 where <lineno> is a decimal line number that could be used by an editor to find <identifier> in  
82488 <filename>.

82489 Neither alternative format shall be produced by *ctags* when it is used as described by  
82490 POSIX.1-2008, but the standard utilities that process tags files shall be able to process those  
82491 formats as well as the first format.

82492 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in  
82493 the POSIX locale.

#### 82494 EXTENDED DESCRIPTION

82495 If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output  
82496 line for each of the following objects:

- 82497 • Function definitions
- 82498 • Type definitions
- 82499 • Macros with arguments

82500 It may also produce output for any of the following objects:

- 82501 • Function prototypes
- 82502 • Structures
- 82503 • Unions
- 82504 • Global variable definitions
- 82505 • Enumeration types
- 82506 • Macros without arguments
- 82507 • **#define** statements
- 82508 • **#line** statements

82509 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C  
82510 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the  
82511 trailing **.c**, and leading pathname components (if any) removed.

82512 On systems that do not support the C-Language Development Utilities option, *ctags* produces  
82513 unspecified results for C-language source code files. It should write to standard error a message  
82514 identifying this condition and cause a non-zero exit status to be produced.

82515 If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each  
82516 function definition. It may also produce output for any of the following objects:

- 82517 • Subroutine definitions

**ctags***Utilities*

- 82518 • COMMON statements
- 82519 • PARAMETER statements
- 82520 • DATA and BLOCK DATA statements
- 82521 • Statement numbers

82522 On systems that do not support the FORTRAN Development Utilities option, *ctags* produces  
 82523 unspecified results for FORTRAN source code files. It should write to standard error a message  
 82524 identifying this condition and cause a non-zero exit status to be produced.

82525 It is implementation-defined what other objects (including duplicate identifiers) produce output.

**EXIT STATUS**

82526 The following exit values shall be returned:

- 82528 0 Successful completion.
- 82529 >0 An error occurred.

**CONSEQUENCES OF ERRORS**

82530 Default.

**APPLICATION USAGE**

82532 The output with `-x` is meant to be a simple index that can be written out as an off-line readable  
 82533 function index. If the input files to *ctags* (such as `.c` files) were not created using the same locale  
 82534 as that in effect when *ctags* `-x` is run, results might not be as expected.

82536 The description of C-language processing says “attempts to” because the C language can be  
 82537 greatly confused, especially through the use of `#defines`, and this utility would be of no use if  
 82538 the real C preprocessor were run to identify them. The output from *ctags* may be fooled and  
 82539 incorrect for various constructs.

**EXAMPLES**

82540 None.

**RATIONALE**

82542 The option list was significantly reduced from that provided by historical implementations. The  
 82543 `-F` option was omitted as redundant, since it is the default. The `-B` option was omitted as being  
 82544 of very limited usefulness. The `-t` option was omitted since the recognition of `typedefs` is now  
 82545 required for C source files. The `-u` option was omitted because the update function was judged  
 82546 to be not only inefficient, but also rarely needed.

82548 An early proposal included a `-w` option to suppress warning diagnostics. Since the types of such  
 82549 diagnostics could not be described, the option was omitted as being not useful.

82550 The text for `LC_CTYPE` about compatibility with the C locale acknowledges that the ISO C  
 82551 standard imposes requirements on the locale used to process C source. This could easily be a  
 82552 superset of that known as “the C locale” by way of implementation extensions, or one of a few  
 82553 alternative locales for systems supporting different codesets. No statement is made for  
 82554 FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar  
 82555 locale concept. However, a general rule in this volume of POSIX.1-2008 is that any time that  
 82556 locales do not match (preparing a file for one locale and processing it in another), the results are  
 82557 suspect.

82558 The collation sequence of the tags file is not affected by `LC_COLLATE` because it is typically not  
 82559 used by human readers, but only by programs such as *vi* to locate the tag within the source files.  
 82560 Using the POSIX locale eliminates some of the problems of coordinating locales between the  
 82561 *ctags* file creator and the *vi* file reader.

82562 Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file  
 82563 has been published to encourage other programs to use the tags in new ways. The format allows  
 82564 either patterns or line numbers to find the identifiers because the historical *vi* recognizes either.  
 82565 The *ctags* utility does not produce the format using line numbers because it is not useful  
 82566 following any source file changes that add or delete lines. The documented search patterns  
 82567 match historical practice. It should be noted that literal leading `<circumflex>` or trailing `<dollar-`  
 82568 `sign>` characters in the search pattern will only behave correctly if anchored to the beginning of  
 82569 the line or end of the line by an additional `<circumflex>` or `<dollar-sign>` character.

82570 Historical implementations also understand the objects used by the languages Pascal and  
 82571 sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is not  
 82572 required to accommodate these languages, although implementors are encouraged to do so.

82573 The following historical option was not specified, as *vgrind* is not included in this volume of  
 82574 POSIX.1-2008:

82575 **-v** If the `-v` flag is given, an index of the form expected by *vgrind* is produced on the  
 82576 standard output. This listing contains the function name, filename, and page  
 82577 number (assuming 64-line pages). Since the output is sorted into lexicographic  
 82578 order, it may be desired to run the output through *sort -f*. Sample use:

```
82579 ctags -v files | sort -f > index vgrind -x index
```

82580 The special treatment of the tag **main** makes the use of *ctags* practical in directories with more  
 82581 than one program.

#### 82582 FUTURE DIRECTIONS

82583 None.

#### 82584 SEE ALSO

82585 [c99](#), [fort77](#), [vi](#)

82586 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 82587 CHANGE HISTORY

82588 First released in Issue 4.

#### 82589 Issue 5

82590 The FUTURE DIRECTIONS section is added.

#### 82591 Issue 6

82592 This utility is marked as part of the User Portability Utilities option.

82593 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

82594 The normative text is reworded to avoid use of the term “must” for application requirements.

82595 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the  
 82596 DESCRIPTION.

#### 82597 Issue 7

82598 The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

82599 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**cut**

Utilities

82600 **NAME**82601 **cut** — cut out selected fields of each line of a file82602 **SYNOPSIS**82603 **cut** *-b list* [*-n*] [*file...*]82604 **cut** *-c list* [*file...*]82605 **cut** *-f list* [*-d delim*] [*-s*] [*file...*]82606 **DESCRIPTION**82607 The *cut* utility shall cut out bytes (*-b* option), characters (*-c* option), or character-delimited fields  
82608 (*-f* option) from each line in one or more files, concatenate them, and write them to standard  
82609 output.82610 **OPTIONS**82611 The *cut* utility shall conform to XBD Section 12.2 (on page 215).82612 The application shall ensure that the option-argument *list* (see options *-b*, *-c*, and *-f* below) is a  
82613 <comma>-separated list or <blank>-separated list of positive numbers and ranges. Ranges can  
82614 be in three forms. The first is two positive numbers separated by a <hyphen> (*low-high*), which  
82615 represents all fields from the first number to the second number. The second is a positive  
82616 number preceded by a <hyphen> (*-high*), which represents all fields from field number 1 to that  
82617 number. The third is a positive number followed by a <hyphen> (*low-*), which represents that  
82618 number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be  
82619 specified in any order, but the bytes, characters, or fields selected shall be written in the order of  
82620 the input data. If an element appears in the selection list more than once, it shall be written  
82621 exactly once.

82622 The following options shall be supported:

82623 **-b list** Cut based on a *list* of bytes. Each selected byte shall be output unless the *-n* option  
82624 is also specified. It shall not be an error to select bytes not present in the input line.82625 **-c list** Cut based on a *list* of characters. Each selected character shall be output. It shall  
82626 not be an error to select characters not present in the input line.82627 **-d delim** Set the field delimiter to the character *delim*. The default is the <tab>.82628 **-f list** Cut based on a *list* of fields, assumed to be separated in the file by a delimiter  
82629 character (see *-d*). Each selected field shall be output. Output fields shall be  
82630 separated by a single occurrence of the field delimiter character. Lines with no field  
82631 delimiters shall be passed through intact, unless *-s* is specified. It shall not be an  
82632 error to select fields not present in the input line.82633 **-n** Do not split characters. When specified with the *-b* option, each element in *list* of  
82634 the form *low-high* (<hyphen>-separated numbers) shall be modified as follows:

- 82635
- If the byte selected by *low* is not the first byte of a character, *low* shall be  
82636 decremented to select the first byte of the character originally selected by *low*.  
82637 If the byte selected by *high* is not the last byte of a character, *high* shall be  
82638 decremented to select the last byte of the character prior to the character  
82639 originally selected by *high*, or zero if there is no prior character. If the  
82640 resulting range element has *high* equal to zero or *low* greater than *high*, the list  
82641 element shall be dropped from *list* for that input line without causing an  
82642 error.

82643 Each element in *list* of the form *low-* shall be treated as above with *high* set to the  
82644 number of bytes in the current line, not including the terminating <newline>. Each

- 82645 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each  
 82646 element in *list* of the form *num* (a single number) shall be treated as above with *low*  
 82647 set to *num* and *high* set to *num*.
- 82648 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless  
 82649 specified, lines with no delimiters shall be passed through untouched.
- 82650 **OPERANDS**
- 82651 The following operand shall be supported:
- 82652 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
 82653 '*-*', the standard input shall be used.
- 82654 **STDIN**
- 82655 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '*-*'.  
 82656 See the INPUT FILES section.
- 82657 **INPUT FILES**
- 82658 The input files shall be text files, except that line lengths shall be unlimited.
- 82659 **ENVIRONMENT VARIABLES**
- 82660 The following environment variables shall affect the execution of *cut*:
- 82661 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 82662 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 82663 variables used to determine the values of locale categories.)
- 82664 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 82665 internationalization variables.
- 82666 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 82667 characters (for example, single-byte as opposed to multi-byte characters in  
 82668 arguments and input files).
- 82669 *LC\_MESSAGES*
- 82670 Determine the locale that should be used to affect the format and contents of  
 82671 diagnostic messages written to standard error.
- 82672 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 82673 **ASYNCHRONOUS EVENTS**
- 82674 Default.
- 82675 **STDOUT**
- 82676 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of  
 82677 the following):
- 82678 "%s\n", <concatenation of bytes>
- 82679 "%s\n", <concatenation of characters>
- 82680 "%s\n", <concatenation of fields and field delimiters>
- 82681 **STDERR**
- 82682 The standard error shall be used only for diagnostic messages.
- 82683 **OUTPUT FILES**
- 82684 None.

## 82685 EXTENDED DESCRIPTION

82686 None.

## 82687 EXIT STATUS

82688 The following exit values shall be returned:

82689 0 All input files were output successfully.

82690 &gt;0 An error occurred.

## 82691 CONSEQUENCES OF ERRORS

82692 Default.

## 82693 APPLICATION USAGE

82694 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.  
 82695 The *cut* utility should be used when the number of lines (or records) needs to remain constant.  
 82696 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

82697 Earlier versions of the *cut* utility worked in an environment where bytes and characters were  
 82698 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In  
 82699 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option  
 82700 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The  
 82701 algorithm specified for **-n** guarantees that:

82702 `cut -b 1-500 -n file > file1`82703 `cut -b 501- -n file > file2`

82704 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,  
 82705 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

## 82706 EXAMPLES

82707 Examples of the option qualifier list:

82708 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

82709 1-3,8 Equivalent to 1,2,3,8.

82710 -5,10 Equivalent to 1,2,3,4,5,10.

82711 3- Equivalent to third to last, inclusive.

82712 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte  
 82713 characters; see the description of **-n**.

82714 The following command:

82715 `cut -d : -f 1,6 /etc/passwd`

82716 reads the System V password file (user database) and produces lines of the form:

82717 `<user ID>:<home directory>`

82718 Most utilities in this volume of POSIX.1-2008 work on text files. The *cut* utility can be used to  
 82719 turn files with arbitrary line lengths into a set of text files containing the same data. The *paste*  
 82720 utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**  
 82721 contains long lines:

82722 `cut -b 1-500 -n file > file1`82723 `cut -b 501- -n file > file2`

82724 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
 82725 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in

82726 **file** that are longer than 500 + {LINE\_MAX} bytes.) The original file can be recreated from **file1**  
82727 and **file2** using the command:

```
82728 paste -d "\0" file1 file2 > file
```

#### 82729 RATIONALE

82730 Some historical implementations do not count <backspace> characters in determining character  
82731 counts with the **-c** option. This may be useful for using *cut* for processing *nroff* output. It was  
82732 deliberately decided not to have the **-c** option treat either <backspace> or <tab> characters in  
82733 any special fashion. The *fold* utility does treat these characters specially.

82734 Unlike other utilities, some historical implementations of *cut* exit after not finding an input file,  
82735 rather than continuing to process the remaining *file* operands. This behavior is prohibited by this  
82736 volume of POSIX.1-2008, where only the exit status is affected by this problem.

82737 The behavior of *cut* when provided with either mutually-exclusive options or options that do  
82738 not work logically together has been deliberately left unspecified in favor of global wording in  
82739 [Section 1.4](#) (on page 2288).

82740 The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The  
82741 change represents historical practice on all known systems. The original standard was  
82742 ambiguous on the nature of the output.

82743 The *list* option-arguments are historically used to select the portions of the line to be written, but  
82744 do not affect the order of the data. For example:

```
82745 echo abcdefghi | cut -c6,2,4-7,1
```

82746 yields "abdefg".

82747 A proposal to enhance *cut* with the following option:

82748 **-o** Preserve the selected field order. When this option is specified, each byte, character, or field  
82749 (or ranges of such) shall be written in the order specified by the *list* option-argument, even if  
82750 this requires multiple outputs of the same bytes, characters, or fields.

82751 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft  
82752 standard.

#### 82753 FUTURE DIRECTIONS

82754 None.

#### 82755 SEE ALSO

82756 [Section 2.5](#) (on page 2301), [fold](#), [grep](#), [paste](#)

82757 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 82758 CHANGE HISTORY

82759 First released in Issue 2.

#### 82760 Issue 6

82761 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

82762 The normative text is reworded to avoid use of the term "must" for application requirements.

#### 82763 Issue 7

82764 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82765 SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE.

**cxref**

Utilities

82766 **NAME**82767 `cxref` — generate a C-language program cross-reference table (**DEVELOPMENT**)82768 **SYNOPSIS**

```
82769 xSI cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
82770 [-U name]... file...
```

82771 **DESCRIPTION**

82772 The `cxref` utility shall analyze a collection of C-language *files* and attempt to build a cross-  
 82773 reference table. Information from **#define** lines shall be included in the symbol table. A sorted  
 82774 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*  
 82775 separately, or with the `-c` option, in combination. Each symbol shall contain an <asterisk> before  
 82776 the declaring reference.

82777 **OPTIONS**

82778 The `cxref` utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the `-D`,  
 82779 `-I`, and `-U` options (which are identical to their interpretation by `c99`) is significant. The  
 82780 following options shall be supported:

- 82781 `-c` Write a combined cross-reference of all input files.
- 82782 `-s` Operate silently; do not print input filenames.
- 82783 `-o file` Direct output to named *file*.
- 82784 `-w num` Format output no wider than *num* (decimal) columns. This option defaults to 80 if  
 82785 *num* is not specified or is less than 51.
- 82786 `-D` Equivalent to `c99`.
- 82787 `-I` Equivalent to `c99`.
- 82788 `-U` Equivalent to `c99`.

82789 **OPERANDS**

82790 The following operand shall be supported:

- 82791 *file* A pathname of a C-language source file.

82792 **STDIN**

82793 Not used.

82794 **INPUT FILES**

82795 The input files are C-language source files.

82796 **ENVIRONMENT VARIABLES**

82797 The following environment variables shall affect the execution of `cxref`:

- 82798 `LANG` Provide a default value for the internationalization variables that are unset or null.  
 82799 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 82800 variables used to determine the values of locale categories.)
- 82801 `LC_ALL` If set to a non-empty string value, override the values of all the other  
 82802 internationalization variables.
- 82803 `LC_COLLATE`  
 82804 Determine the locale for the ordering of the output.
- 82805 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
 82806 characters (for example, single-byte as opposed to multi-byte characters in  
 82807 arguments and input files).

- 82808 *LC\_MESSAGES*
- 82809 Determine the locale that should be used to affect the format and contents of
- 82810 diagnostic messages written to standard error.
- 82811 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 82812 **ASYNCHRONOUS EVENTS**
- 82813 Default.
- 82814 **STDOUT**
- 82815 The standard output shall be used for the cross-reference listing, unless the `-o` option is used to
- 82816 select a different output file.
- 82817 The format of standard output is unspecified, except that the following information shall be
- 82818 included:
- 82819 • If the `-c` option is not specified, each portion of the listing shall start with the name of the
  - 82820 input file on a separate line.
  - 82821 • The name line shall be followed by a sorted list of symbols, each with its associated
  - 82822 location pathname, the name of the function in which it appears (if it is not a function
  - 82823 name itself), and line number references.
  - 82824 • Each line number may be preceded by an <asterisk> (`'*'` ) flag, meaning that this is the
  - 82825 declaring reference. Other single-character flags, with implementation-defined meanings,
  - 82826 may be included.
- 82827 **STDERR**
- 82828 The standard error shall be used only for diagnostic messages.
- 82829 **OUTPUT FILES**
- 82830 The output file named by the `-o` option shall be used instead of standard output.
- 82831 **EXTENDED DESCRIPTION**
- 82832 None.
- 82833 **EXIT STATUS**
- 82834 The following exit values shall be returned:
- 82835 0 Successful completion.
  - 82836 >0 An error occurred.
- 82837 **CONSEQUENCES OF ERRORS**
- 82838 Default.
- 82839 **APPLICATION USAGE**
- 82840 None.
- 82841 **EXAMPLES**
- 82842 None.
- 82843 **RATIONALE**
- 82844 None.
- 82845 **FUTURE DIRECTIONS**
- 82846 None.

82847 **SEE ALSO**82848 [c99](#)82849 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)82850 **CHANGE HISTORY**

82851 First released in Issue 2.

82852 **Issue 5**82853 In the SYNOPSIS, [-U *dir*] is changed to [-U *name*].82854 **Issue 6**

82855 The APPLICATION USAGE section is added.

82856 **Issue 7**

82857 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## Utilities

## date

## 82858 NAME

82859 date — write the date and time

## 82860 SYNOPSIS

82861 date [-u] [+format]

82862 XSI date [-u] mmddhhmm[[cc]yy]

## 82863 DESCRIPTION

82864 XSI The *date* utility shall write the date and time to standard output or attempt to set the system  
 82865 date and time. By default, the current date and time shall be written. If an operand beginning  
 82866 with '+' is specified, the output format of *date* shall be controlled by the conversion  
 82867 specifications and other text in the operand.

## 82868 OPTIONS

82869 The *date* utility shall conform to XBD Section 12.2 (on page 215).

82870 The following option shall be supported:

82871 **-u** Perform operations as if the *TZ* environment variable was set to the string "UTC0",  
 82872 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone  
 82873 indicated by the *TZ* environment variable or the system default if that variable is  
 82874 unset or null.

## 82875 OPERANDS

82876 The following operands shall be supported:

82877 **+format** When the format is specified, each conversion specifier shall be replaced in the  
 82878 standard output by its corresponding value. All other characters shall be copied to  
 82879 the output without change. The output shall always be terminated with a  
 82880 <newline>.

## 82881 Conversion Specifications

82882 %a Locale's abbreviated weekday name.  
 82883 %A Locale's full weekday name.  
 82884 %b Locale's abbreviated month name.  
 82885 %B Locale's full month name.  
 82886 %c Locale's appropriate date and time representation.  
 82887 %C Century (a year divided by 100 and truncated to an integer) as a decimal  
 82888 number [00,99].  
 82889 %d Day of the month as a decimal number [01,31].  
 82890 %D Date in the format *mm/dd/yy*.  
 82891 %e Day of the month as a decimal number [1,31] in a two-digit field with  
 82892 leading <space> character fill.  
 82893 %h A synonym for %b.  
 82894 %H Hour (24-hour clock) as a decimal number [00,23].  
 82895 %I Hour (12-hour clock) as a decimal number [01,12].

|       |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 82896 | %j  | Day of the year as a decimal number [001,366].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 82897 | %m  | Month as a decimal number [01,12].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 82898 | %M  | Minute as a decimal number [00,59].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 82899 | %n  | A <newline>.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82900 | %p  | Locale's equivalent of either AM or PM.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 82901 | %r  | 12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 82902 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82903 | %S  | Seconds as a decimal number [00,60].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82904 | %t  | A <tab>.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 82905 | %T  | 24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 82906 | %u  | Weekday as a decimal number [1,7] (1=Monday).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 82907 | %U  | Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.                                                                                                                                                                                                                                                                                                                                                                |
| 82908 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82909 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82910 | %V  | Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.                                                                                                                                                                                                                                             |
| 82911 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82912 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82913 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82914 | %w  | Weekday as a decimal number [0,6] (0=Sunday).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 82915 | %W  | Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.                                                                                                                                                                                                                                                                                                                                                                |
| 82916 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82917 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82918 | %x  | Locale's appropriate date representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82919 | %X  | Locale's appropriate time representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82920 | %y  | Year within century [00,99].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82921 | %Y  | Year with century as a decimal number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 82922 | %Z  | Timezone name, or no characters if no timezone is determinable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82923 | %%  | A <percent-sign> character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 82924 |     | See XBD <a href="#">Section 7.3.5</a> (on page 158) for the conversion specifier values in the POSIX locale.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82925 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82926 |     | <b>Modified Conversion Specifications</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82927 |     | Some conversion specifiers can be modified by the <b>E</b> and <b>O</b> modifier characters to indicate a different format or specification as specified in the <i>LC_TIME</i> locale description (see XBD <a href="#">Section 7.3.5</a> , on page 158). If the corresponding keyword (see <b>era</b> , <b>era_year</b> , <b>era_d_fmt</b> , and <b>alt_digits</b> in XBD <a href="#">Section 7.3.5</a> , on page 158) is not specified or not supported for the current locale, the unmodified conversion specifier value shall be used. |
| 82928 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82929 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82930 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82931 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82932 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 82933 | %Ec | Locale's alternative appropriate date and time representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Utilities

## date

|       |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 82934 | %EC          | The name of the base year (period) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 82935 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82936 | %Ex          | Locale's alternative date representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 82937 | %EX          | Locale's alternative time representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 82938 | %Ey          | Offset from %EC (year only) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82939 | %EY          | Full alternative year representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 82940 | %Od          | Day of month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 82941 | %Oe          | Day of month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 82942 | %OH          | Hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82943 | %OI          | Hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82944 | %Om          | Month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 82945 | %OM          | Minutes using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82946 | %OS          | Seconds using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 82947 | %Ou          | Weekday as a number in the locale's alternative representation (Monday = 1).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 82948 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82949 | %OU          | Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 82950 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82951 | %OV          | Week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 82952 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82953 | %Ow          | Weekday as a number in the locale's alternative representation (Sunday = 0).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 82954 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82955 | %OW          | Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 82956 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82957 | %Oy          | Year (offset from %C) in alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 82958 | XSI          | <code>mmddhhmm[[cc]yy]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 82959 |              | Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if <i>yy</i> is omitted. |
| 82960 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82961 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82962 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82963 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82964 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82965 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82966 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82967 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82968 | <b>Note:</b> | It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 82969 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82970 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82971 | <b>STDIN</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 82972 |              | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

**date**

Utilities

82973 **INPUT FILES**

82974 None.

82975 **ENVIRONMENT VARIABLES**82976 The following environment variables shall affect the execution of *date*:

82977 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 82978 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 82979 variables used to determine the values of locale categories.)

82980 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 82981 internationalization variables.

82982 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 82983 characters (for example, single-byte as opposed to multi-byte characters in  
 82984 arguments).

82985 *LC\_MESSAGES*

82986 Determine the locale that should be used to affect the format and contents of  
 82987 diagnostic messages written to standard error.

82988 *LC\_TIME* Determine the format and contents of date and time strings written by *date*.

82989 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

82990 *TZ* Determine the timezone in which the time and date are written, unless the *-u*  
 82991 option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an  
 82992 unspecified system default timezone is used.

82993 **ASYNCHRONOUS EVENTS**

82994 Default.

82995 **STDOUT**

82996 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to  
 82997 specifying:

82998 `date "+%a %b %e %H:%M:%S %Z %Y"`

82999 **STDERR**

83000 The standard error shall be used only for diagnostic messages.

83001 **OUTPUT FILES**

83002 None.

83003 **EXTENDED DESCRIPTION**

83004 None.

83005 **EXIT STATUS**

83006 The following exit values shall be returned:

83007 0 The date was written successfully.

83008 &gt;0 An error occurred.

83009 **CONSEQUENCES OF ERRORS**

83010 Default.

83011 **APPLICATION USAGE**

83012 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can  
83013 contain <newline> characters in some locales, so it may be difficult to use the format shown in  
83014 standard output for parsing the output of *date* in those locales.

83015 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap  
83016 second.

83017 Although certain of the conversion specifiers in the POSIX locale (such as the name of the  
83018 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
83019 using these fields may need to adjust the capitalization if the output is going to be used at the  
83020 beginning of a sentence.

83021 The date string formatting capabilities are intended for use in Gregorian-style calendars,  
83022 possibly with a different starting year (or years). The %x and %c conversion specifications,  
83023 however, are intended for local representation; these may be based on a different, non-Gregorian  
83024 calendar.

83025 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year  
83026 format base year); it can be viewed as the base of the current subdivision in the Gregorian  
83027 calendar. The century number is calculated as the year divided by 100 and truncated to an  
83028 integer; it should not be confused with the use of ordinal numbers for centuries (for example,  
83029 "twenty-first century".) Both the %EY and %y can then be viewed as the offset from %EC and %C,  
83030 respectively.

83031 The E and O modifiers modify the traditional conversion specifiers, so that they can always be  
83032 used, even if the implementation (or the current locale) does not support the modifier.

83033 The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as  
83034 these are based on the Gregorian calendar system. Extending the E modifiers to other date  
83035 elements may provide an implementation-defined extension capable of supporting other  
83036 calendar systems, especially in combination with the O modifier.

83037 The O modifier supports time and date formats using the locale's alternative numerical symbols,  
83038 such as Kanji or Hindi digits or ordinal number representation.

83039 Non-European locales, whether they use Latin digits in computational items or not, often have  
83040 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a  
83041 variant of dates uses Roman numerals for the months: the third day of September 1991 would be  
83042 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking  
83043 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion  
83044 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O  
83045 modifier was introduced to support the use for display purposes of non-Latin digits. In the  
83046 LC\_TIME category in *localedef*, the optional **alt\_digits** keyword is intended for this purpose. As  
83047 an example, assume the following (partial) *localedef* source:

```
83048 alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
83049            "IX";"X";"XI";"XII"
83050 d_fmt      "%e.%Om.%Y"
```

83051 With the above date, the command:

```
83052 date "+%x"
```

83053 would yield 3.IX.1991. With the same **d\_fmt**, but without the **alt\_digits**, the command would  
83054 yield 3.9.1991.

## 83055 EXAMPLES

83056 1. The following are input/output examples of *date* used at arbitrary times in the POSIX  
83057 locale:

```
83058 $ date
83059 Tue Jun 26 09:58:10 PDT 1990

83060 $ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
83061 DATE: 11/02/91
83062 TIME: 13:36:16
```

```
83063 $ date "+TIME: %r"
83064 TIME: 01:36:32 PM
```

83065 2. Examples for Denmark, where the default date and time format is `%a %d %b %Y %T %Z`:

```
83066 $ LANG=da_DK.iso_8859-1 date
83067 ons 02 okt 1991 15:03:32 CET

83068 $ LANG=da_DK.iso_8859-1 \
83069 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
83070 DATO: onsdag den 2. oktober 1991
83071 KLOKKEN: 15:03:56
```

83072 3. Examples for Germany, where the default date and time format is `%a %d.%h.%Y, %T %Z`:

```
83073 $ LANG=De_DE.88591 date
83074 Mi 02.Okt.1991, 15:01:21 MEZ

83075 $ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
83076 DATUM: Mittwoch, 02. Oktober 1991
83077 ZEIT: 15:02:02
```

83078 4. Examples for France, where the default date and time format is `%a %d %h %Y %Z %T`:

```
83079 $ LANG=Fr_FR.88591 date
83080 Mer 02 oct 1991 MET 15:03:32

83081 $ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
83082 JOUR: Mercredi 02 octobre 1991
83083 HEURE: 15:03:56
```

## 83084 RATIONALE

83085 Some of the new options for formatting are from the ISO C standard. The `-u` option was  
83086 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is  
83087 allowed as an equivalent TZ value to be compatible with all of the systems using the BSD  
83088 implementation, where this option originated.

83089 The `%e` format conversion specification (adopted from System V) was added because the ISO C  
83090 standard conversion specifications did not provide any way to produce the historical default  
83091 *date* output during the first nine days of any month.

83092 There are two varieties of day and week numbering supported (in addition to any others created  
83093 with the locale-dependent `%E` and `%O` modifier characters):

- 83094 • The historical variety in which Sunday is the first day of the week and the weekdays  
83095 preceding the first Sunday of the year are considered week 0. These are represented by `%w`  
83096 and `%U`. A variant of this is `%W`, using Monday as the first day of the week, but still  
83097 referring to week 0. This view of the calendar was retained because so many historical

- 83098 applications depend on it and the ISO C standard *strftime()* function, on which many *date*  
83099 implementations are based, was defined in this way.
- 83100 • The international standard, based on the ISO 8601:2004 standard where Monday is the first  
83101 weekday and the algorithm for the first week number is more complex: If the week  
83102 (Monday to Sunday) containing January 1 has four or more days in the new year, then it is  
83103 week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These  
83104 are represented by the new conversion specifications %u and %V, added as a result of  
83105 international comments.
- 83106 **FUTURE DIRECTIONS**
- 83107 None.
- 83108 **SEE ALSO**
- 83109 XBD [Section 7.3.5](#) (on page 158), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 83110 XSH [fprintf\(\)](#), [strftime\(\)](#)
- 83111 **CHANGE HISTORY**
- 83112 First released in Issue 2.
- 83113 **Issue 5**
- 83114 Changes are made for Year 2000 alignment.
- 83115 **Issue 6**
- 83116 The following new requirements on POSIX implementations derive from alignment with the  
83117 Single UNIX Specification:
- 83118 • The %EX modified conversion specification is added.
- 83119 The Open Group Corrigendum U048/2 is applied, correcting the examples.
- 83120 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors  
83121 for consistency with the *LC\_TIME* category.
- 83122 A clarification is made such that the current year is the default if the *yy* argument is omitted  
83123 when setting the system date and time.
- 83124 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE  
83125 HISTORY section.

**dd**

Utilities

83126 **NAME**

83127 dd — convert and copy a file

83128 **SYNOPSIS**83129 dd [*operand*. . .]83130 **DESCRIPTION**

83131 The *dd* utility shall copy the specified input file to the specified output file with possible  
 83132 conversions using specific input and output block sizes. It shall read the input one block at a  
 83133 time, using the specified input block size; it shall then process the block of data actually  
 83134 returned, which could be smaller than the requested block size. It shall apply any conversions  
 83135 that have been specified and write the resulting data to the output in blocks of the specified  
 83136 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,  
 83137 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a  
 83138 separate output block; if the read returns less than a full block and the **sync** conversion is not  
 83139 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**  
 83140 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the  
 83141 input shall be processed and collected into full-sized output blocks until the end of the input is  
 83142 reached.

83143 The processing order shall be as follows:

- 83144 1. An input block is read.
- 83145 2. If the input block is shorter than the specified input block size and the **sync** conversion is  
 83146 specified, null bytes shall be appended to the input data up to the specified size. (If either  
 83147 **block** or **unblock** is also specified, <space> characters shall be appended instead of null  
 83148 bytes.) The remaining conversions and output shall include the pad characters as if they  
 83149 had been read from the input.
- 83150 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is  
 83151 requested, the resulting data shall be written to the output as a single block, and the  
 83152 remaining steps are omitted.
- 83153 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If  
 83154 there is an odd number of bytes in the input block, the last byte in the input record shall  
 83155 not be swapped.
- 83156 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These  
 83157 conversions shall operate on the input data independently of the input blocking; an input  
 83158 or output fixed-length record may span block boundaries.
- 83159 6. The data resulting from input or conversion or both shall be aggregated into output  
 83160 blocks of the specified size. After the end of input is reached, any remaining output shall  
 83161 be written as a block without padding if **conv=sync** is not specified; thus, the final output  
 83162 block may be shorter than the output block size.

83163 **OPTIONS**

83164 None.

83165 **OPERANDS**83166 All of the operands shall be processed before any input is read. The following operands shall be  
 83167 supported:

- 83168 **if=file** Specify the input pathname; the default is standard input.
- 83169 **of=file** Specify the output pathname; the default is standard output. If the **seek=expr**  
 83170 conversion is not also specified, the output file shall be truncated before the copy  
 83171 begins if an explicit **of=file** operand is specified, unless **conv=notrunc** is specified.

|       |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 83172 |                               | If <b>seek=expr</b> is specified, but <b>conv=notrunc</b> is not, the effect of the copy shall be to preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy. If the input file is empty and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist, the size of the output file shall be set to the file offset after the seek.)                          |
| 83173 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83174 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83175 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83176 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83177 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83178 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83179 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83180 | <b>ibs=expr</b>               | Specify the input block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 83181 | <b>obs=expr</b>               | Specify the output block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 83182 | <b>bs=expr</b>                | Set both input and output block sizes to <i>expr</i> bytes, superseding <b>ibs=</b> and <b>obs=</b> . If no conversion other than <b>sync</b> , <b>noerror</b> , and <b>notrunc</b> is specified, each input block shall be copied to the output as a single block without aggregating short blocks.                                                                                                                                                                                                                                                                                                                                                               |
| 83183 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83184 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83185 | <b>cbs=expr</b>               | Specify the conversion block size for <b>block</b> and <b>unblock</b> in bytes by <i>expr</i> (default is zero). If <b>cbs=</b> is omitted or given a value of zero, using <b>block</b> or <b>unblock</b> produces unspecified results.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 83186 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83187 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83188 | XSI                           | The application shall ensure that this operand is also specified if the <b>conv=</b> operand is specified with a value of <b>ascii</b> , <b>ebcdic</b> , or <b>ibm</b> . For a <b>conv=</b> operand with an <b>ascii</b> value, the input is handled as described for the <b>unblock</b> value, except that characters are converted to ASCII before any trailing <space> characters are deleted. For <b>conv=</b> operands with <b>ebcdic</b> or <b>ibm</b> values, the input is handled as described for the <b>block</b> value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space> characters are added. |
| 83189 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83190 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83191 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83192 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83193 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83194 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83195 | <b>skip=n</b>                 | Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 83196 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83197 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83198 | <b>seek=n</b>                 | Skip <i>n</i> blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.                                                                                                                                                                                                                                                           |
| 83199 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83200 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83201 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83202 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83203 | <b>count=n</b>                | Copy only <i>n</i> input blocks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 83204 | <b>conv=value[,value ...]</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83205 |                               | Where <i>values</i> are <comma>-separated symbols from the following list:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 83206 | XSI                           | <b>ascii</b> Convert EBCDIC to ASCII; see Table 4-7 (on page 2585).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 83207 | XSI                           | <b>ebcdic</b> Convert ASCII to EBCDIC; see Table 4-7 (on page 2585).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 83208 | XSI                           | <b>ibm</b> Convert ASCII to a different EBCDIC set; see Table 4-8 (on page 2586).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 83209 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83210 | XSI                           | The <b>ascii</b> , <b>ebcdic</b> , and <b>ibm</b> values are mutually-exclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 83211 | <b>block</b>                  | Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space> characters shall be appended to lines that are shorter than their conversion block size to fill the block.                                                                                                                                                                                                                                   |
| 83212 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83213 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83214 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83215 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 83216 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|       |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 83217 |                | Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the <code>STDERR</code> section).                                                                                                                                                                                                                                                                         |
| 83218 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83219 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83220 |                | The <b>block</b> and <b>unblock</b> values are mutually-exclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 83221 | <b>unblock</b> | Convert fixed-length records to variable length. Read a number of bytes equal to the conversion block size (or the number of bytes remaining in the input, if less than the conversion block size), delete all trailing <space> characters, and append a <newline>.                                                                                                                                                                                                                                 |
| 83222 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83223 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83224 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83225 | <b>lcase</b>   | Map uppercase characters specified by the <code>LC_CTYPE</code> keyword <b>tolower</b> to the corresponding lowercase character. Characters for which no mapping is specified shall not be modified by this conversion.                                                                                                                                                                                                                                                                             |
| 83226 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83227 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83228 |                | The <b>lcase</b> and <b>ucase</b> symbols are mutually-exclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 83229 | <b>ucase</b>   | Map lowercase characters specified by the <code>LC_CTYPE</code> keyword <b>toupper</b> to the corresponding uppercase character. Characters for which no mapping is specified shall not be modified by this conversion.                                                                                                                                                                                                                                                                             |
| 83230 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83231 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83232 | <b>swab</b>    | Swap every pair of input bytes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83233 | <b>noerror</b> | Do not stop processing on an input error. When an input error occurs, a diagnostic message shall be written on standard error, followed by the current input and output block counts in the same format as used at completion (see the <code>STDERR</code> section). If the <b>sync</b> conversion is specified, the missing input shall be replaced with null bytes and processed normally; otherwise, the input block shall be omitted from the output.                                           |
| 83234 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83235 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83236 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83237 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83238 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83239 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83240 | <b>notrunc</b> | Do not truncate the output file. Preserve blocks in the output file not explicitly written by this invocation of the <code>dd</code> utility. (See also the preceding <code>of=file</code> operand.)                                                                                                                                                                                                                                                                                                |
| 83241 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83242 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83243 | <b>sync</b>    | Pad every input block to the size of the <code>ibs=</code> buffer, appending null bytes. (If either <b>block</b> or <b>unblock</b> is also specified, append <space> characters, rather than null bytes.)                                                                                                                                                                                                                                                                                           |
| 83244 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83245 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83246 |                | The behavior is unspecified if operands other than <code>conv=</code> are specified more than once.                                                                                                                                                                                                                                                                                                                                                                                                 |
| 83247 |                | For the <code>bs=</code> , <code>cbs=</code> , <code>ibs=</code> , and <code>obs=</code> operands, the application shall supply an expression specifying a size in bytes. The expression, <i>expr</i> , can be:                                                                                                                                                                                                                                                                                     |
| 83248 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83249 |                | 1. A positive decimal number                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 83250 |                | 2. A positive decimal number followed by <i>k</i> , specifying multiplication by 1 024                                                                                                                                                                                                                                                                                                                                                                                                              |
| 83251 |                | 3. A positive decimal number followed by <i>b</i> , specifying multiplication by 512                                                                                                                                                                                                                                                                                                                                                                                                                |
| 83252 |                | 4. Two or more positive decimal numbers (with or without <i>k</i> or <i>b</i> ) separated by <i>x</i> , specifying the product of the indicated values                                                                                                                                                                                                                                                                                                                                              |
| 83253 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83254 |                | All of the operands are processed before any input is read.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 83255 | XSI            | The following two tables display the octal number character values used for the <b>ascii</b> and <b>ebcdic</b> conversions (first table) and for the <b>ibm</b> conversion (second table). In both tables, the ASCII values are the row and column headers and the EBCDIC values are found at their intersections. For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one |
| 83256 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83257 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83258 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 83259 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

83260  
83261

correspondence with these tables. The differences between the two tables are highlighted by small boxes drawn around five entries.

83262

Table 4-7 ASCII to EBCDIC Conversion

|      | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| 0000 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL |
| 0001 |     | HT  | LF  | VT  | FF  | CR  | SO  | SI  |
| 0002 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB |
| 0003 | CAN | EM  | SUB | ESC | IFS | IGS | IRS | ITB |
| 0004 | Sp  | !   | "   | #   | \$  | %   | &   | '   |
| 0005 | (   | )   | *   | +   | ,   | -   | .   | /   |
| 0006 | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| 0007 | 8   | 9   | :   | ;   | <   | =   | >   | ?   |
| 0008 | @   | A   | B   | C   | D   | E   | F   | G   |
| 0009 | H   | I   | J   | K   | L   | M   | N   | O   |
| 0010 | P   | Q   | R   | S   | T   | U   | V   | W   |
| 0011 | X   | Y   | Z   | [   | \   | ]   | ^   | _   |
| 0012 | ,   | a   | b   | c   | d   | e   | f   | g   |
| 0013 | h   | i   | j   | k   | l   | m   | n   | o   |
| 0014 | p   | q   | r   | s   | t   | u   | v   | w   |
| 0015 | x   | y   | z   | {   |     | }   | ~   | DEL |
| 0016 | DS  | SOS | FS  | WUS | BYP | NL  | RNL | POC |
| 0017 | SA  | SFE | SM  | CSP | MFA | SPS | RPT | CU1 |
| 0018 |     |     | UBS | IR  | PP  | TRN | NBS | GE  |
| 0019 | SBS | IT  | RFF | CU3 | SEL | RES |     |     |
| 0020 |     |     |     |     |     |     |     |     |
| 0021 |     |     |     |     |     |     |     |     |
| 0022 |     |     |     |     |     |     |     |     |
| 0023 |     |     |     |     |     |     |     |     |
| 0024 |     |     |     |     |     |     |     |     |
| 0025 |     |     |     |     |     |     |     |     |
| 0026 |     |     |     |     |     |     |     |     |
| 0027 |     |     |     |     |     |     |     |     |
| 0028 |     |     |     |     |     |     |     |     |
| 0029 |     |     |     |     |     |     |     |     |
| 0030 |     |     |     |     |     |     |     |     |
| 0031 |     |     |     |     |     |     |     |     |
| 0032 |     |     |     |     |     |     |     |     |
| 0033 |     |     |     |     |     |     |     |     |
| 0034 |     |     |     |     |     |     |     |     |
| 0035 |     |     |     |     |     |     |     |     |
| 0036 |     |     |     |     |     |     |     |     |
| 0037 |     |     |     |     |     |     |     |     |

83263

Table 4-8 ASCII to IBM EBCDIC Conversion

|      | 0        | 1        | 2        | 3        | 4        | 5        | 6        | 7        |
|------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0000 | 0000 NUL | 0001 SOH | 0002 STX | 0003 ETX | 0067 EOT | 0055 ENQ | 0056 ACK | 0057 BEL |
| 0010 | 0026 BS  | 0005 HT  | 0045 LF  | 0013 VT  | 0014 FF  | 0015 CR  | 0016 SO  | 0017 SI  |
| 0020 | 0020 DLE | 0021 DC1 | 0022 DC2 | 0023 DC3 | 0074 DC4 | 0075 NAK | 0062 SYN | 0046 ETB |
| 0030 | 0030 CAN | 0031 EM  | 0077 SUB | 0047 ESC | 0034 IFS | 0035 IGS | 0036 IRS | 0037 ITB |
| 0040 | 0100 Sp  | 0132 !   | 0177 "   | 0173 #   | 0133 \$  | 0154 %   | 0120 &   | 0175 .   |
| 0050 | 0115 (   | 0135 )   | 0134 *   | 0116 +   | 0153 ,   | 0140 -   | 0113 /   | 0141 /   |
| 0060 | 0360 0   | 0361 1   | 0362 2   | 0363 3   | 0364 4   | 0365 5   | 0366 6   | 0367 7   |
| 0070 | 0370 8   | 0371 9   | 0172 :   | 0136 ;   | 0114 <   | 0176 =   | 0156 >   | 0157 ?   |
| 0100 | 0174 @   | 0301 A   | 0302 B   | 0303 C   | 0304 D   | 0305 E   | 0306 F   | 0307 G   |
| 0110 | 0310 H   | 0311 I   | 0321 J   | 0322 K   | 0323 L   | 0324 M   | 0325 N   | 0326 O   |
| 0120 | 0327 P   | 0330 Q   | 0331 R   | 0342 S   | 0343 T   | 0344 U   | 0345 V   | 0346 W   |
| 0130 | 0347 X   | 0350 Y   | 0351 Z   | 0255 [   | 0340 \   | 0275 ]   | 0137 ^   | 0155 _   |
| 0140 | 0171 `   | 0201 a   | 0202 b   | 0203 c   | 0204 d   | 0205 e   | 0206 f   | 0207 g   |
| 0150 | 0210 h   | 0211 i   | 0221 j   | 0222 k   | 0223 l   | 0224 m   | 0225 n   | 0226 o   |
| 0160 | 0227 p   | 0230 q   | 0231 r   | 0242 s   | 0243 t   | 0244 u   | 0245 v   | 0246 w   |
| 0170 | 0247 x   | 0250 y   | 0251 z   | 0300 {   | 0117     | 0320 }   | 0241 DEL | 0007 DEL |
| 0200 | 0040 DS  | 0041 SOS | 0042 FS  | 0043 WUS | 0044 BYP | 0025 NL  | 0006 RNL | 0027 POC |
| 0210 | 0050 SA  | 0051 SFE | 0052 SM  | 0053 CSP | 0054 MFA | 0011 SPS | 0012 RPT | 0033 CU1 |
| 0220 | 0060     | 0061     | 0032 UBS | 0063 IR  | 0064 PP  | 0065 TRN | 0066 NBS | 0010 GE  |
| 0230 | 0070 SBS | 0071 IT  | 0072 RFF | 0073 CU3 | 0004 SEL | 0024 RES | 0076     | 0341     |
| 0240 | 0101     | 0102     | 0103     | 0104     | 0105     | 0106     | 0107     | 0110     |
| 0250 | 0111     | 0121     | 0122     | 0123     | 0124     | 0125     | 0126     | 0127     |
| 0260 | 0130     | 0131     | 0142     | 0143     | 0144     | 0145     | 0146     | 0147     |
| 0270 | 0150     | 0151     | 0160     | 0161     | 0162     | 0163     | 0164     | 0165     |
| 0300 | 0166     | 0167     | 0170     | 0200     | 0212     | 0213     | 0214     | 0215     |
| 0310 | 0216     | 0217     | 0220     | 0232     | 0233     | 0234     | 0235     | 0236     |
| 0320 | 0237     | 0240     | 0252     | 0253     | 0254     | 0255     | 0256     | 0257     |
| 0330 | 0260     | 0261     | 0262     | 0263     | 0264     | 0265     | 0266     | 0267     |
| 0340 | 0270     | 0271     | 0272     | 0273     | 0274     | 0275     | 0276     | 0277     |
| 0350 | 0312     | 0313     | 0314     | 0315     | 0316     | 0317     | 0332     | 0333     |
| 0360 | 0334     | 0335     | 0336     | 0337     | 0352     | 0353     | 0354     | 0355     |
| 0370 | 0356     | 0357     | 0372     | 0373     | 0374     | 0375     | 0376     | 0377 EO  |

83264 **STDIN**

83265 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

83266 **INPUT FILES**

83267 The input file can be any file type.

83268 **ENVIRONMENT VARIABLES**

83269 The following environment variables shall affect the execution of *dd*:

83270 **LANG** Provide a default value for the internationalization variables that are unset or null.  
83271 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
83272 variables used to determine the values of locale categories.)

83273 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
83274 internationalization variables.

83275 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
83276 characters (for example, single-byte as opposed to multi-byte characters in  
83277 arguments and input files), the classification of characters as uppercase or  
83278 lowercase, and the mapping of characters from one case to the other.

83279 **LC\_MESSAGES**

83280 Determine the locale that should be used to affect the format and contents of  
83281 diagnostic messages written to standard error and informative messages written to  
83282 standard output.

83283 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

83284 **ASYNCHRONOUS EVENTS**

83285 For **SIGINT**, the *dd* utility shall interrupt its current processing, write status information to  
83286 standard error, and exit as though terminated by **SIGINT**. It shall take the standard action for all  
83287 other signals; see the ASYNCHRONOUS EVENTS section in Section 1.4 (on page 2288).

83288 **STDOUT**

83289 If no **of=** operand is specified, the standard output shall be used. The nature of the output  
83290 depends on the operands selected.

83291 **STDERR**

83292 On completion, *dd* shall write the number of input and output blocks to standard error. In the  
83293 POSIX locale the following formats shall be used:

83294 "%u+%u records in\n", <number of whole input blocks>,  
83295 <number of partial input blocks>

83296 "%u+%u records out\n", <number of whole output blocks>,  
83297 <number of partial output blocks>

83298 A partial input block is one for which *read()* returned less than the input block size. A partial  
83299 output block is one that was written with fewer bytes than specified by the output block size.

83300 In addition, when there is at least one truncated block, the number of truncated blocks shall be  
83301 written to standard error. In the POSIX locale, the format shall be:

83302 "%u truncated %s\n", <number of truncated blocks>, "record" (if  
83303 <number of truncated blocks> is one) "records" (otherwise)

83304 Diagnostic messages may also be written to standard error.

83305 **OUTPUT FILES**

83306 If the **of=** operand is used, the output shall be the same as described in the **STDOUT** section.

83307 **EXTENDED DESCRIPTION**

83308 None.

83309 **EXIT STATUS**

83310 The following exit values shall be returned:

83311 0 The input file was copied successfully.

83312 >0 An error occurred.

83313 **CONSEQUENCES OF ERRORS**

83314 If an input error is detected and the **noerror** conversion has not been specified, any partial  
83315 output block shall be written to the output file, a diagnostic message shall be written, and the  
83316 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall  
83317 be written and the copy operation shall be discontinued.

83318 **APPLICATION USAGE**

83319 The input and output block size can be specified to take advantage of raw physical I/O.

83320 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions  
83321 specified for the *dd* utility perform conversions for the version specified by the tables.

83322 **EXAMPLES**

83323 The following command:

```
83324 dd if=/dev/rmt0h of=/dev/rmt1h
```

83325 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

83326 The following command:

```
83327 dd ibs=10 skip=1
```

83328 strips the first 10 bytes from standard input.

83329 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the  
83330 ASCII file *x*:

```
83331 dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

83332 **RATIONALE**

83333 The **OPTIONS** section is listed as “None” because there are no options recognized by historical  
83334 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax  
83335 Guidelines, which would have resulted in the classic hyphenated option letters. In this version  
83336 of this volume of POSIX.1-2008, *dd* retains its curious JCL-like syntax due to the large number of  
83337 applications that depend on the historical implementation.

83338 A suggested implementation technique for **conv=noerror,sync** is to zero (or <space>-fill, if  
83339 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input  
83340 buffer to the output even after an error. In this manner, any data transferred to the input buffer  
83341 before the error was detected is preserved. Another point is that a failed read on a regular file or  
83342 a disk generally does not increment the file offset, and *dd* must then seek past the block on which  
83343 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic  
83344 tape, however, the tape normally has passed the block containing the error when the error is  
83345 reported, and thus no seek is necessary.

83346 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely  
83347 portable) scripts that assume these values. If they were left unspecified, unusual results could

- 83348 occur if an implementation chose an odd block size.
- 83349 Historical implementations of *dd* used *creat()* when processing **of=file**. This makes the **seek=**  
83350 operand unusable except on special files. The **conv=notrunc** feature was added because more  
83351 recent BSD-based implementations use *open()* (without `O_TRUNC`) instead of *creat()*, but they  
83352 fail to delete output file contents after the data copied.
- 83353 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to  
83354 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of  
83355 POSIX.1-2008.
- 83356 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are  
83357 taken from a common print train that does contain them. Other than those characters, the print  
83358 train values are not filled in, but appear to provide some of the motivation for the historical  
83359 choice of translations reflected here.
- 83360 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.
- 83361 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ  
83362 in such a way that:
- 83363 1. EBCDIC 0112 ('ϕ') and 0152 (broken pipe) do not appear in the table.
  - 83364 2. EBCDIC 0137 ('↵') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC  
83365 0232 (no graphic) is used.
  - 83366 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC  
83367 0137 ('↵') is used.
  - 83368 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table  
83369 and once in place of 0112 ('ϕ') and 0241 ('~').
- 83370 In net result:
- 83371 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.
- 83372 That displaced EBCDIC 0137 ('↵') in cell 0176.
- 83373 That displaced EBCDIC 0232 (no graphic) in cell 0136.
- 83374 That replaced EBCDIC 0152 (broken pipe) in cell 0313.
- 83375 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('ϕ').
- 83376 This translation, however, reflects historical practice that (ASCII) '^' and '↵' were often  
83377 mapped to each other, as were '[' and 'ϕ'; and ']' and (EBCDIC) '~'.
- 83378 The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the  
83379 **ascii** operand, the input is handled as described for the **unblock** operand except that characters  
83380 are converted to ASCII before the trailing <space> characters are deleted. For the **ebcdic** and  
83381 **ibm** operands, the input is handled as described for the **block** operand except that the characters  
83382 are converted to EBCDIC or IBM EBCDIC after the trailing <space> characters are added.
- 83383 The **block** and **unblock** keywords are from historical BSD practice.
- 83384 The consistent use of the word **record** in standard error messages matches most historical  
83385 practice. An earlier version of System V used **block**, but this has been updated in more recent  
83386 releases.
- 83387 Early proposals only allowed two numbers separated by **x** to be used in a product when  
83388 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of

- 83389 allowing multiple numbers in the product as provided by Version 7 and all releases of System V  
83390 and BSD.
- 83391 A change to the **swab** conversion is required to match historical practice and is the result of IEEE  
83392 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 83393 A change to the handling of SIGINT is required to match historical practice and is the result of  
83394 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 83395 **FUTURE DIRECTIONS**
- 83396 None.
- 83397 **SEE ALSO**
- 83398 [Section 1.4](#) (on page 2288), *sed*, *tr*
- 83399 [XBD Chapter 8](#) (on page 173)
- 83400 **CHANGE HISTORY**
- 83401 First released in Issue 2.
- 83402 **Issue 5**
- 83403 The second paragraph of the **cbs=** description is reworded and marked EX.
- 83404 The FUTURE DIRECTIONS section is added.
- 83405 **Issue 6**
- 83406 Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b  
83407 draft standard.
- 83408 The normative text is reworded to avoid use of the term “must” for application requirements.
- 83409 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between **dd of=file** and  
83410 **conv=notrunc**.
- 83411 **Issue 7**
- 83412 Austin Group Interpretation 1003.1-2001 #102 is applied.
- 83413 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

## 83414 NAME

83415 delta — make a delta (change) to an SCCS file (DEVELOPMENT)

## 83416 SYNOPSIS

83417 xSI delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...

## 83418 DESCRIPTION

83419 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that  
83420 were made to the files retrieved by *get* (called the *g-files*, or generated files).

## 83421 OPTIONS

83422 The *delta* utility shall conform to XBD Section 12.2 (on page 215), except that the *-y* option has an  
83423 optional option-argument. This optional option-argument shall not be presented as a separate  
83424 argument.

83425 The following options shall be supported:

83426 **-r SID** Uniquely identify which delta is to be made to the SCCS file. The use of this option  
83427 shall be necessary only if two or more outstanding *get* commands for editing (*get*  
83428 *-e*) on the same SCCS file were done by the same person (login name). The SID  
83429 value specified with the *-r* option can be either the SID specified on the *get*  
83430 command line or the SID to be made as reported by the *get* utility; see *get* (on page  
83431 2764).83432 **-s** Suppress the report to standard output of the activity associated with each *file*. See  
83433 the STDOUT section.83434 **-n** Specify retention of the edited *g-file* (normally removed at completion of delta  
83435 processing).83436 **-g list** Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when  
83437 the file is accessed at the change level (SID) created by this delta.83438 **-m mrlist** Specify a modification request (MR) number that the application shall supply as  
83439 the reason for creating the new delta. This shall be used if the SCCS file has the *v*  
83440 flag set; see *admin*.83441 If *-m* is not used and *'-'* is not specified as a file argument, and the standard  
83442 input is a terminal, the prompt described in the STDOUT section shall be written  
83443 to standard output before the standard input is read; if the standard input is not a  
83444 terminal, no prompt shall be issued.83445 MRs in a list shall be separated by <blank> characters or escaped <newline>  
83446 characters. An unescaped <newline> shall terminate the MR list. The escape  
83447 character is <backslash>.83448 If the *v* flag has a value, it shall be taken to be the name of a program which  
83449 validates the correctness of the MR numbers. If a non-zero exit status is returned  
83450 from the MR number validation program, the *delta* utility shall terminate. (It is  
83451 assumed that the MR numbers were not all valid.)83452 **-y[comment]** Describe the reason for making the delta. The *comment* shall be an arbitrary group  
83453 of lines that would meet the definition of a text file. Implementations shall support  
83454 *comments* from zero to 512 bytes and may support longer values. A null string  
83455 (specified as either *-y*, *-y" "*, or in response to a prompt for a comment) shall be  
83456 considered a valid *comment*.83457 If *-y* is not specified and *'-'* is not specified as a file argument, and the standard

**delta**

Utilities

83458 input is a terminal, the prompt described in the STDOUT section shall be written  
 83459 to standard output before the standard input is read; if the standard input is not a  
 83460 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the  
 83461 comment text. The escape character is <backslash>.

83462 The `-y` option shall be required if the *file* operand is specified as `'-'`.

83463 **-p** Write (to standard output) the SCCS file differences before and after the delta is  
 83464 applied in *diff* format; see *diff*.

**OPERANDS**

83465 The following operand shall be supported:

83466 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*  
 83467 utility shall behave as though each file in the directory were specified as a named  
 83468 file, except that non-SCCS files (last component of the pathname does not begin  
 83469 with `s.`) and unreadable files shall be silently ignored.  
 83470

83471 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;  
 83472 each line of the standard input shall be taken to be the name of an SCCS file to be  
 83473 processed. Non-SCCS files and unreadable files shall be silently ignored.

**STDIN**

83474 The standard input shall be a text file used only in the following cases:

- 83475 • To read an *mrlist* or a *comment* (see the `-m` and `-y` options).
- 83476 • A *file* operand shall be specified as `'-'`. In this case, the `-y` option must be used to specify  
 83477 the comment, and if the SCCS file has the `v` flag set, the `-m` option must also be used to  
 83478 specify the MR list.  
 83479

**INPUT FILES**

83480 Input files shall be text files whose data is to be included in the SCCS files. If the first character of  
 83481 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file  
 83482 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be  
 83483 99 999 for this delta.  
 83484

**ENVIRONMENT VARIABLES**

83485 The following environment variables shall affect the execution of *delta*:

83486 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 83487 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 83488 variables used to determine the values of locale categories.)  
 83489

83490 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 83491 internationalization variables.

83492 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 83493 characters (for example, single-byte as opposed to multi-byte characters in  
 83494 arguments and input files).

83495 *LC\_MESSAGES*

83496 Determine the locale that should be used to affect the format and contents of  
 83497 diagnostic messages written to standard error, and informative messages written  
 83498 to standard output.

83499 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

83500 **TZ** Determine the timezone in which the time and date are written in the SCCS file. If  
83501 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

### 83502 **ASYNCHRONOUS EVENTS**

83503 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit  
83504 code. The standard action shall be taken for all other signals; see [Section 1.4](#) (on page 2288).

### 83505 **STDOUT**

83506 The standard output shall be used only for the following messages in the POSIX locale:

- 83507 • Prompts (see the *-m* and *-y* options) in the following formats:

83508 "MRs? "

83509 "comments? "

83510 The MR prompt, if written, shall always precede the comments prompt.

- 83511 • A report of each file's activities (unless the *-s* option is specified) in the following format:

83512 "%s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID>,  
83513 <number of lines inserted>, <number of lines deleted>,  
83514 <number of lines unchanged>

### 83515 **STDERR**

83516 The standard error shall be used only for diagnostic messages.

### 83517 **OUTPUT FILES**

83518 Any SCCS files updated shall be files of an unspecified format.

### 83519 **EXTENDED DESCRIPTION**

#### 83520 **System Date and Time**

83521 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new  
83522 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the  
83523 behavior is unspecified.

### 83524 **EXIT STATUS**

83525 The following exit values shall be returned:

83526 0 Successful completion.

83527 >0 An error occurred.

### 83528 **CONSEQUENCES OF ERRORS**

83529 Default.

### 83530 **APPLICATION USAGE**

83531 Problems can arise if the system date and time have been modified (for example, put forward  
83532 and then back again, or unsynchronized clocks across a network) and can also arise when  
83533 different values of the *TZ* environment variable are used.

83534 Problems of a similar nature can also arise for the operation of the *get* utility, which records the  
83535 date and time in the file body.

### 83536 **EXAMPLES**

83537 None.

|       |                                                                                                                                                                                                                                                                                 |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 83538 | <b>RATIONALE</b>                                                                                                                                                                                                                                                                |
| 83539 | None.                                                                                                                                                                                                                                                                           |
| 83540 | <b>FUTURE DIRECTIONS</b>                                                                                                                                                                                                                                                        |
| 83541 | None.                                                                                                                                                                                                                                                                           |
| 83542 | <b>SEE ALSO</b>                                                                                                                                                                                                                                                                 |
| 83543 | Section 1.4 (on page 2288), <i>admin, diff, get, prs, rmdel</i>                                                                                                                                                                                                                 |
| 83544 | XBD Chapter 8 (on page 173), Section 12.2 (on page 215)                                                                                                                                                                                                                         |
| 83545 | <b>CHANGE HISTORY</b>                                                                                                                                                                                                                                                           |
| 83546 | First released in Issue 2.                                                                                                                                                                                                                                                      |
| 83547 | <b>Issue 5</b>                                                                                                                                                                                                                                                                  |
| 83548 | The output format description in the STDOUT section is corrected.                                                                                                                                                                                                               |
| 83549 | <b>Issue 6</b>                                                                                                                                                                                                                                                                  |
| 83550 | The APPLICATION USAGE section is added.                                                                                                                                                                                                                                         |
| 83551 | The normative text is reworded to avoid use of the term “must” for application requirements.                                                                                                                                                                                    |
| 83552 | The Open Group Base Resolution bwg2001-007 is applied as follows:                                                                                                                                                                                                               |
| 83553 | • The use of ‘-’ as a file argument is clarified.                                                                                                                                                                                                                               |
| 83554 | • The use of STDIN is added.                                                                                                                                                                                                                                                    |
| 83555 | • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that implementations re-signal themselves when catching a normally fatal signal.                                                                                                                |
| 83556 |                                                                                                                                                                                                                                                                                 |
| 83557 | • New text is added to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.                                                                                                                                                                   |
| 83558 |                                                                                                                                                                                                                                                                                 |
| 83559 | New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the TZ environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007. |
| 83560 |                                                                                                                                                                                                                                                                                 |
| 83561 |                                                                                                                                                                                                                                                                                 |
| 83562 |                                                                                                                                                                                                                                                                                 |
| 83563 | <b>Issue 7</b>                                                                                                                                                                                                                                                                  |
| 83564 | SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.                                                                                                                                                                                                                               |

83565 **NAME**

83566 df — report free disk space

83567 **SYNOPSIS**83568 XSI df [-k] [-P|-t] [*file*...]83569 **DESCRIPTION**

83570 XSI The *df* utility shall write the amount of available space and file slots for file systems on which  
 83571 the invoking user has appropriate read access. File systems shall be specified by the *file*  
 83572 operands; when none are specified, information shall be written for all file systems. The format  
 83573 of the default output from *df* is unspecified, but all space figures are reported in 512-byte units,  
 83574 unless the *-k* option is specified. This output shall contain at least the file system names, amount  
 83575 XSI of available space on each of these file systems, and the number of free file slots, or *inodes*,  
 83576 available; when *-t* is specified, the output shall contain the total allocated space as well.

83577 **OPTIONS**83578 The *df* utility shall conform to XBD Section 12.2 (on page 215).

83579 The following options shall be supported:

83580 *-k* Use 1 024-byte units, instead of the default 512-byte units, when writing space  
 83581 figures.83582 *-P* Produce output in the format described in the STDOUT section.83583 XSI *-t* Include total allocated-space figures in the output.83584 **OPERANDS**

83585 The following operand shall be supported:

83586 *file* A pathname of a file within the hierarchy of the desired file system. If a file other than a FIFO, a regular file, a directory, or a special file representing the device  
 83587 XSI containing the file system (for example, */dev/dsk/0s1*) is specified, the results are  
 83588 unspecified. If the *file* operand names a file other than a special file containing a file  
 83589 system, *df* shall write the amount of free space in the file system containing the  
 83590 specified *file* operand. Otherwise, *df* shall write the amount of free space in that  
 83591 XSI file system.  
 83592

83593 **STDIN**

83594 Not used.

83595 **INPUT FILES**

83596 None.

83597 **ENVIRONMENT VARIABLES**83598 The following environment variables shall affect the execution of *df*:

83599 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 83600 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 83601 variables used to determine the values of locale categories.)

83602 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 83603 internationalization variables.

83604 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 83605 characters (for example, single-byte as opposed to multi-byte characters in  
 83606 arguments).

- 83607 *LC\_MESSAGES*
- 83608 Determine the locale that should be used to affect the format and contents of
- 83609 diagnostic messages written to standard error and informative messages written to
- 83610 standard output.
- 83611 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 83612 **ASYNCHRONOUS EVENTS**
- 83613 Default.
- 83614 **STDOUT**
- 83615 When both the *-k* and *-P* options are specified, the following header line shall be written (in the
- 83616 POSIX locale):
- 83617 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"
- 83618 When the *-P* option is specified without the *-k* option, the following header line shall be written
- 83619 (in the POSIX locale):
- 83620 "Filesystem 512-blocks Used Available Capacity Mounted on\n"
- 83621 The implementation may adjust the spacing of the header line and the individual data lines so
- 83622 that the information is presented in orderly columns.
- 83623 The remaining output with *-P* shall consist of one line of information for each specified file
- 83624 system. These lines shall be formatted as follows:
- 83625 "%s %d %d %d %d%% %s\n", *<file system name>*, *<total space>*,
- 83626 *<space used>*, *<space free>*, *<percentage used>*,
- 83627 *<file system root>*
- 83628 In the following list, all quantities expressed in 512-byte units (1 024-byte when *-k* is specified)
- 83629 shall be rounded up to the next higher unit. The fields are:
- 83630 *<file system name>*
- 83631 The name of the file system, in an implementation-defined format.
- 83632 *<total space>* The total size of the file system in 512-byte units. The exact meaning of this figure
- 83633 is implementation-defined, but should include *<space used>*, *<space free>*, plus any
- 83634 space reserved by the system not normally available to a user.
- 83635 *<space used>* The total amount of space allocated to existing files in the file system, in 512-byte
- 83636 units.
- 83637 *<space free>* The total amount of space available within the file system for the creation of new
- 83638 files by unprivileged users, in 512-byte units. When this figure is less than or equal
- 83639 to zero, it shall not be possible to create any new files on the file system without
- 83640 first deleting others, unless the process has appropriate privileges. The figure
- 83641 written may be less than zero.
- 83642 *<percentage used>*
- 83643 The percentage of the normally available space that is currently allocated to all files
- 83644 on the file system. This shall be calculated using the fraction:
- 83645 
$$\frac{\textit{<space used>}}{\textit{<space used> + <space free>}}$$
- 83646 expressed as a percentage. This percentage may be greater than 100 if *<space free>*
- 83647 is less than zero. The percentage value shall be expressed as a positive integer, with
- 83648 any fractional result causing it to be rounded to the next highest integer.

- 83649 <*file system root*>  
 83650 The directory below which the file system hierarchy appears.
- 83651 XSI The output format is unspecified when `-t` is used.
- 83652 **STDERR**  
 83653 The standard error shall be used only for diagnostic messages.
- 83654 **OUTPUT FILES**  
 83655 None.
- 83656 **EXTENDED DESCRIPTION**  
 83657 None.
- 83658 **EXIT STATUS**  
 83659 The following exit values shall be returned:
- 83660 0 Successful completion.  
 83661 >0 An error occurred.
- 83662 **CONSEQUENCES OF ERRORS**  
 83663 Default.
- 83664 **APPLICATION USAGE**  
 83665 On most systems, the “name of the file system, in an implementation-defined format” is the  
 83666 special file on which the file system is mounted.
- 83667 On large file systems, the calculation specified for percentage used can create huge rounding  
 83668 errors.
- 83669 **EXAMPLES**
- 83670 1. The following example writes portable information about the `/usr` file system:  
 83671 `df -P /usr`
- 83672 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same  
 83673 output as the previous example:  
 83674 `df -P /usr/src`
- 83675 **RATIONALE**  
 83676 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase  
 83677 `-P` was selected to avoid collision with a known industry extension using `-p`.
- 83678 Historical `df` implementations vary considerably in their default output. It was therefore  
 83679 necessary to describe the default output in a loose manner to accommodate all known historical  
 83680 implementations and to add a portable option (`-P`) to provide information in a portable format.
- 83681 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other  
 83682 utilities in this volume of POSIX.1-2008. This does not mandate that the file system itself be  
 83683 based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed by  
 83684 the standard developers that 512 bytes was the best default unit because of its complete  
 83685 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and  
 83686 that a `-k` option to switch to 1 024-byte units was a good compromise. Users who prefer the  
 83687 more logical 1 024-byte quantity can easily alias `df` to `df -k` without breaking many historical  
 83688 scripts relying on the 512-byte units.
- 83689 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`  
 83690 environment variable to achieve consistency and user acceptance. Since this is not historical

**df**

- 83691 practice on any system, it is left as a possible area for system extensions and will be re-evaluated  
83692 in a future version if it is widely implemented.
- 83693 **FUTURE DIRECTIONS**  
83694 None.
- 83695 **SEE ALSO**  
83696 *find*  
83697 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 83698 **CHANGE HISTORY**  
83699 First released in Issue 2.
- 83700 **Issue 6**  
83701 This utility is marked as part of the User Portability Utilities option.
- 83702 **Issue 7**  
83703 Austin Group Interpretation 1003.1-2001 #099 is applied.  
83704 The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is  
83705 now an option for interactive utilities.  
83706 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

83707 **NAME**

83708 diff — compare two files

83709 **SYNOPSIS**

83710 diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2

83711 **DESCRIPTION**

83712 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of  
 83713 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be  
 83714 produced if the files are identical.

83715 **OPTIONS**83716 The *diff* utility shall conform to XBD Section 12.2 (on page 215).

83717 The following options shall be supported:

83718 **-b** Cause any amount of white space at the end of a line to be treated as a single  
 83719 <newline> (that is, the white-space characters preceding the <newline> are  
 83720 ignored) and other strings of white-space characters, not including <newline>  
 83721 characters, to compare equal.

83722 **-c** Produce output in a form that provides three lines of copied context.

83723 **-C n** Produce output in a form that provides *n* lines of copied context (where *n* shall be  
 83724 interpreted as a positive decimal integer).

83725 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be used  
 83726 to convert *file1* into *file2*.

83727 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to  
 83728 be suitable as input for the *ed* utility, and in the opposite order.

83729 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*  
 83730 are both directories.

83731 The *diff* utility shall detect infinite loops; that is, entering a previously visited  
 83732 directory that is an ancestor of the last file encountered. When it detects an infinite  
 83733 loop, *diff* shall write a diagnostic message to standard error and shall either recover  
 83734 its position in the hierarchy or terminate.

83735 **-u** Produce output in a form that provides three lines of unified context.

83736 **-U n** Produce output in a form that provides *n* lines of unified context (where *n* shall be  
 83737 interpreted as a non-negative decimal integer).

83738 **OPERANDS**

83739 The following operands shall be supported:

83740 *file1, file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the  
 83741 standard input shall be used in its place.

83742 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special  
 83743 files, or FIFO special files to any files and shall not compare regular files to directories. Further  
 83744 details are as specified in Diff Directory Comparison Format (on page 2600). The behavior of *diff*  
 83745 on other file types is implementation-defined when found in directories.

83746 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file  
 83747 contained in the directory file with a filename that is the same as the last component of the non-  
 83748 directory file.

**diff**

- 83749 **STDIN**  
 83750 The standard input shall be used only if one of the *file1* or *file2* operands references standard  
 83751 input. See the INPUT FILES section.
- 83752 **INPUT FILES**  
 83753 The input files may be of any type.
- 83754 **ENVIRONMENT VARIABLES**  
 83755 The following environment variables shall affect the execution of *diff*:
- 83756 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 83757 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 83758 variables used to determine the values of locale categories.)
- 83759 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 83760 internationalization variables.
- 83761 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 83762 characters (for example, single-byte as opposed to multi-byte characters in  
 83763 arguments and input files).
- 83764 *LC\_MESSAGES*  
 83765 Determine the locale that should be used to affect the format and contents of  
 83766 diagnostic messages written to standard error and informative messages written to  
 83767 standard output.
- 83768 *LC\_TIME* Determine the locale for affecting the format of file timestamps written with the *-C*  
 83769 and *-c* options.
- 83770 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 83771 *TZ* Determine the timezone used for calculating file timestamps written with a context  
 83772 format. If *TZ* is unset or null, an unspecified default timezone shall be used.
- 83773 **ASYNCHRONOUS EVENTS**  
 83774 Default.
- 83775 **STDOUT**
- 83776 **Diff Directory Comparison Format**  
 83777 If both *file1* and *file2* are directories, the following output formats shall be used.  
 83778 In the POSIX locale, each file that is present in only one directory shall be reported using the  
 83779 following format:  
 83780 "Only in %s: %s\n", <directory pathname>, <filename>  
 83781 In the POSIX locale, subdirectories that are common to the two directories may be reported with  
 83782 the following format:  
 83783 "Common subdirectories: %s and %s\n", <directory1 pathname>,  
 83784 <directory2 pathname>  
 83785 For each file common to the two directories, if the two files are not to be compared: if the two  
 83786 files have the same device ID and file serial number, or are both block special files that refer to  
 83787 the same device, or are both character special files that refer to the same device, in the POSIX  
 83788 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format  
 83789 shall be used that contains the pathnames of the two files.  
 83790 For each file common to the two directories, if the files are compared and are identical, no

83791 output shall be written. If the two files differ, the following format is written:

83792 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>

83793 where <diff\_options> are the options as specified on the command line.

83794 All directory pathnames listed in this section shall be relative to the original command line  
83795 arguments. All other names of files listed in this section shall be filenames (pathname  
83796 components).

#### 83797 **Diff Binary Output Format**

83798 In the POSIX locale, if one or both of the files being compared are not text files, it is  
83799 implementation-defined whether *diff* uses the binary file output format or the other formats as  
83800 specified below. The binary file output format shall contain the pathnames of two files being  
83801 compared and the string "differ".

83802 If both files being compared are text files, depending on the options specified, one of the  
83803 following formats shall be used to write the differences.

#### 83804 **Diff Default Output Format**

83805 The default (without *-e*, *-f*, *-c*, *-C*, *-u*, or *-U* options) *diff* utility output shall contain lines of  
83806 these forms:

83807 "%da%d\n", <num1>, <num2>

83808 "%da%d,%d\n", <num1>, <num2>, <num3>

83809 "%dd%d\n", <num1>, <num2>

83810 "%d,%dd%d\n", <num1>, <num2>, <num3>

83811 "%dc%d\n", <num1>, <num2>

83812 "%d,%dc%d\n", <num1>, <num2>, <num3>

83813 "%dc%d,%d\n", <num1>, <num2>, <num3>

83814 "%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>

83815 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the  
83816 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*  
83817 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in  
83818 *ed*, identical pairs (where *num1*=*num2*) are abbreviated as a single number.

83819 Following each of these lines, *diff* shall write to standard output all lines affected in the first file  
83820 using the format:

83821 "<Δ%s", <line>

83822 and all lines affected in the second file using the format:

83823 ">Δ%s", <line>

83824 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are  
83825 separated with a line consisting of three <hyphen> characters:

83826 "---\n"

83827 **Diff -e Output Format**

83828 With the `-e` option, a script shall be produced that shall, when provided as input to `ed`, along  
 83829 with an appended `w` (write) command, convert *file1* into *file2*. Only the `a` (append), `c` (change), `d`  
 83830 (delete), `i` (insert), and `s` (substitute) commands of `ed` shall be used in this script. Text lines,  
 83831 except those consisting of the single character <period> (`'.'`), shall be output as they appear in  
 83832 the file.

83833 **Diff -f Output Format**

83834 With the `-f` option, an alternative format of script shall be produced. It is similar to that  
 83835 produced by `-e`, with the following differences:

- 83836 1. It is expressed in reverse sequence; the output of `-e` orders changes from the end of the  
 83837 file to the beginning; the `-f` from beginning to end.
- 83838 2. The command form `<lines> <command-letter>` used by `-e` is reversed. For example,  
 83839 `10c` with `-e` would be `c10` with `-f`.
- 83840 3. The form used for ranges of line numbers is `<space>`-separated, rather than  
 83841 `<comma>`-separated.

83842 **Diff -c or -C Output Format**

83843 With the `-c` or `-C` option, the output format shall consist of affected lines along with  
 83844 surrounding lines of context. The affected lines shall show which ones need to be deleted or  
 83845 changed in *file1*, and those added from *file2*. With the `-c` option, three lines of context, if  
 83846 available, shall be written before and after the affected lines. With the `-C` option, the user can  
 83847 specify how many lines of context are written. The exact format follows.

83848 The name and last modification time of each file shall be output in the following format:

```
83849 "*** %s %s\n", file1, <file1 timestamp>
83850 "--- %s %s\n", file2, <file2 timestamp>
```

83851 Each `<file>` field shall be the pathname of the corresponding file being compared. The pathname  
 83852 written for standard input is unspecified.

83853 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following  
 83854 command:

```
83855 date "+%a %b %e %T %Y"
```

83856 without the trailing `<newline>`, executed at the time of last modification of the corresponding  
 83857 file (or the current time, if the file is standard input).

83858 Then, the following output formats shall be applied for every set of changes.

83859 First, a line shall be written in the following format:

```
83860 "*****\n"
```

83861 Next, the range of lines in *file1* shall be written in the following format if the range contains two  
 83862 or more lines:

```
83863 "*** %d,%d ****\n", <beginning line number>, <ending line number>
```

83864 and the following format otherwise:

```
83865 "*** %d ****\n", <ending line number>
```

83866 The ending line number of an empty range shall be the number of the preceding line, or 0 if the

83867 range is at the start of the file.

83868 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected  
83869 lines shall be written in the following format:

83870 " $\Delta\Delta\%s$ ", <unaffected\_line>

83871 Deleted lines shall be written as:

83872 " $-\Delta\%s$ ", <deleted\_line>

83873 Changed lines shall be written as:

83874 " $!\Delta\%s$ ", <changed\_line>

83875 Next, the range of lines in *file2* shall be written in the following format if the range contains two  
83876 or more lines:

83877 " $---\ %d,\%d\ ----\ \backslash n$ ", <beginning line number>, <ending line number>

83878 and the following format otherwise:

83879 " $---\ %d\ ----\ \backslash n$ ", <ending line number>

83880 Then, lines of context and changed lines shall be written as described in the previous formats.  
83881 Lines added from *file2* shall be written in the following format:

83882 " $+\Delta\%s$ ", <added\_line>

83883 **Diff  $-u$  or  $-U$  Output Format**

83884 The  $-u$  or  $-U$  options behave like the  $-c$  or  $-C$  options, except that the context lines are not  
83885 repeated; instead, the context, deleted, and added lines are shown together, interleaved. The  
83886 exact format follows.

83887 The name and last modification time of each file shall be output in the following format:

83888 " $---\ \Delta\%s\%s\%s\ \Delta\%s0$ , *file1*, <file1 timestamp>, <file1 frac>, <file1 zone>  
83889 " $+++ \Delta\%s\%s\%s\ \Delta\%s0$ , *file2*, <file2 timestamp>, <file2 frac>, <file2 zone>

83890 Each <*file*> field shall be the pathname of the corresponding file being compared, or the single  
83891 character ' $-$ ' if standard input is being compared. However, if the pathname contains a <tab>  
83892 or a <newline> or if it does not consist entirely of characters taken from the portable character  
83893 set, the behavior is implementation-defined.

83894 Each <*timestamp*> field shall be equivalent to the output from the following command:

83895 `date '+%Y-%m-%d\ \Delta\%H:\%M:\%S'`

83896 without the trailing <newline>, executed at the time of last modification of the corresponding  
83897 file (or the current time, if the file is standard input).

83898 Each <*frac*> field shall be either empty, or a decimal point followed by at least one decimal digit,  
83899 indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional  
83900 digits shall be at least the number needed to represent the file's timestamp without loss of  
83901 information.

83902 Each <*zone*> field shall be of the form "*shhmm*", where "*shh*" is a signed two-digit decimal  
83903 number in the range  $-24$  through  $+25$ , and "*mm*" is an unsigned two-digit decimal number in the  
83904 range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh)  
83905 and minutes (mm) east (+) or west ( $-$ ) of UTC for the timestamp. If the hours and minutes are  
83906 both zero, the sign shall be ' $+$ '. However, if the timezone is not an integral number of minutes

**diff**

- 83907 away from UTC, the `<zone>` field is implementation-defined.
- 83908 Then, the following output formats shall be applied for every set of changes.
- 83909 First, the range of lines in each file shall be written in the following format:
- 83910 "@@Δ-%sΔ+%sΔ@@", `<file1 range>`, `<file2 range>`
- 83911 Each `<range>` field shall be of the form:
- 83912 "%ld", `<beginning line number>`
- 83913 if the range contains exactly one line, and:
- 83914 "%ld,%ld", `<beginning line number>`, `<number of lines>`
- 83915 otherwise. If a range is empty, its beginning line number shall be the number of the line just before the range, or 0 if the empty range starts the file.
- 83917 Next, the affected lines along with lines of context shall be written. Each non-empty unaffected line shall be written in the following format:
- 83918 "Δ%s", `<unaffected_line>`
- 83919 where the contents of the unaffected line shall be taken from `file1`. It is implementation-defined whether an empty unaffected line is written as an empty line or a line containing a single `<space>` character. This line also represents the same line of `file2`, even though `file2`'s line may contain different contents due to the `-b`. Deleted lines shall be written as:
- 83924 "-%s", `<deleted_line>`
- 83925 Added lines shall be written as:
- 83926 "+%s", `<added_line>`
- 83927 The order of lines written shall be the same as that of the corresponding file. A deleted line shall never be written immediately after an added line.
- 83929 If `-U n` is specified, the output shall contain no more than `n` consecutive unaffected lines; and if the output contains an affected line and this line is adjacent to up to `n` consecutive unaffected lines in the corresponding file, the output shall contain these unaffected lines. `-u` shall act like `-U3`.
- 83933 **STDERR**
- 83934 The standard error shall be used only for diagnostic messages.
- 83935 **OUTPUT FILES**
- 83936 None.
- 83937 **EXTENDED DESCRIPTION**
- 83938 None.
- 83939 **EXIT STATUS**
- 83940 The following exit values shall be returned:
- 83941 0 No differences were found.
- 83942 1 Differences were found.
- 83943 >1 An error occurred.

83944 **CONSEQUENCES OF ERRORS**

83945 Default.

83946 **APPLICATION USAGE**

83947 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a  
 83948 delete and add, as a change, or as a change and add; *diff* is not expected to know which  
 83949 happened and users should not care about the difference in output as long as it clearly shows  
 83950 the differences between the files.

83951 **EXAMPLES**

83952 If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory  
 83953 named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**,  
 83954 the command:

83955 `diff -r dir1 dir2`

83956 could produce output similar to:

```
83957 Common subdirectories: dir1/x and dir2/x
83958 Only in dir2/x: y
83959 diff -r dir1/x/date.out dir2/x/date.out
83960 1c1
83961 < Mon Jul 2 13:12:16 PDT 1990
83962 ----
83963 > Tue Jun 19 21:41:39 PDT 1990
```

83964 **RATIONALE**

83965 The **-h** option was omitted because it was insufficiently specified and does not add to  
 83966 applications portability.

83967 Historical implementations employ algorithms that do not always produce a minimum list of  
 83968 differences; the current language about making every effort is the best this volume of  
 83969 POSIX.1-2008 can do, as there is no metric that could be employed to judge the quality of  
 83970 implementations against any and all file contents. The statement “This list should be minimal”  
 83971 clearly implies that implementations are not expected to provide the following output when  
 83972 comparing two 100-line files that differ in only one character on a single line:

```
83973 1,100c1,100
83974 all 100 lines from file1 preceded with "< "
83975 ----
83976 all 100 lines from file2 preceded with "> "
```

83977 The “Only in” messages required when the **-r** option is specified are not used by most historical  
 83978 implementations if the **-e** option is also specified. It is required here because it provides useful  
 83979 information that must be provided to update a target directory hierarchy to match a source  
 83980 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when  
 83981 the **-r** option is specified. They are allowed here but are not required because they are reporting  
 83982 on something that is the same, not reporting a difference, and are not needed to update a target  
 83983 hierarchy.

83984 The **-c** option, which writes output in a format using lines of context, has been included. The  
 83985 format is useful for a variety of reasons, among them being much improved readability and the  
 83986 ability to understand difference changes when the target file has line numbers that differ from  
 83987 another similar, but slightly different, copy. The *patch* utility is most valuable when working  
 83988 with difference listings using a context format. The BSD version of **-c** takes an optional  
 83989 argument specifying the amount of context. Rather than overloading **-c** and breaking the Utility  
 83990 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for

83991 specifying a context diff with a specified amount of context (`-C`). Also, the format for context  
 83992 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines  
 83993 from each other to be merged together. The output format contains an additional four `<asterisk>`  
 83994 characters after the range of affected lines in the first filename. This was to provide a flag for old  
 83995 programs (like old versions of *patch*) that only understand the old context format. The version of  
 83996 context described here does not require that multiple changes within context lines be merged,  
 83997 but it does not prohibit it either. The extension is upwards-compatible, so any vendors that wish  
 83998 to retain the old version of *diff* can do so by adding the extra four `<asterisk>` characters (that is,  
 83999 utilities that currently use *diff* and understand the new merged format will also understand the  
 84000 old unmerged format, but not *vice versa*).

84001 The `-u` and `-U` options of GNU *diff* have been included. Their output format, designed by  
 84002 Wayne Davison, takes up less space than `-c` and `-C` format, and in many cases is easier to read.  
 84003 The format's timestamps do not vary by locale, so `LC_TIME` does not affect it. The format's line  
 84004 numbers are rendered with the `%1d` format, not `%d`, because the file format notation rules would  
 84005 allow extra `<blank>` characters to appear around the numbers.

84006 The substitute command was added as an additional format for the `-e` option. This was added  
 84007 to provide implementations with a way to fix the classic "dot alone on a line" bug present in  
 84008 many versions of *diff*. Since many implementations have fixed this bug, the standard developers  
 84009 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing  
 84010 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then  
 84011 terminate the append command with a period, and then use the substitute command to convert  
 84012 the two periods into one period.

84013 The BSD-derived `-r` option was added to provide a mechanism for using *diff* to compare two file  
 84014 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not  
 84015 easily reproducible with the *find* utility.

84016 The requirement that *diff* not compare files in some circumstances, even though they have the  
 84017 same name, is based on the actual output of historical implementations. The specified behavior  
 84018 precludes the problems arising from running into FIFOs and other files that would cause *diff* to  
 84019 hang waiting for input with no indication to the user that *diff* was hung. An earlier version of  
 84020 this standard specified the output format more precisely, but in practice this requirement was  
 84021 widely ignored and the benefit of standardization seemed small, so it is now unspecified. In  
 84022 most common usage, *diff -r* should indicate differences in the file hierarchies, not the difference  
 84023 of contents of devices pointed to by the hierarchies.

84024 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of  
 84025 POSIX.1-2008 supports named pipes, the standard developers decided that such a restriction  
 84026 was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

84027 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in  
 84028 that it prints out all of the differences at the current level, including the statements about all  
 84029 common subdirectories before recursing into those subdirectories.

84030 The message:

84031 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

84032 does not vary by locale because it is the representation of a command, not an English sentence.

#### 84033 FUTURE DIRECTIONS

84034 None.

84035 **SEE ALSO**84036 *cmp, comm, ed, find*

84037 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

84038 **CHANGE HISTORY**

84039 First released in Issue 2.

84040 **Issue 5**

84041 The FUTURE DIRECTIONS section is added.

84042 **Issue 6**84043 The following new requirements on POSIX implementations derive from alignment with the  
84044 Single UNIX Specification:

- 84045
- The `-f` option is added.

84046 The output format for `-c` or `-C` format is changed to align with changes to the IEEE P1003.2b  
84047 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

84048 The normative text is reworded to avoid use of the term “must” for application requirements.

84049 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT  
84050 section. This changes the specification of *diff -c* so that it agrees with existing practice when  
84051 contexts contain zero lines or one line.84052 **Issue 7**

84053 Austin Group Interpretations 1003.1-2001 #115 and #114 are applied.

84054 Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if both files are  
84055 non-text files.

84056 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84057 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.

**dirname**

Utilities

84058 **NAME**

84059           dirname — return the directory portion of a pathname

84060 **SYNOPSIS**84061           dirname *string*84062 **DESCRIPTION**

84063           The *string* operand shall be treated as a pathname, as defined in XBD Section 3.266 (on page 75).  
 84064           The string *string* shall be converted to the name of the directory containing the filename  
 84065           corresponding to the last pathname component in *string*, performing actions equivalent to the  
 84066           following steps in order:

- 84067           1. If *string* is //, skip steps 2 to 5.
- 84068           2. If *string* consists entirely of <slash> characters, *string* shall be set to a single <slash>  
 84069           character. In this case, skip steps 3 to 8.
- 84070           3. If there are any trailing <slash> characters in *string*, they shall be removed.
- 84071           4. If there are no <slash> characters remaining in *string*, *string* shall be set to a single  
 84072           <period> character. In this case, skip steps 5 to 8.
- 84073           5. If there are any trailing non-<slash> characters in *string*, they shall be removed.
- 84074           6. If the remaining *string* is //, it is implementation-defined whether steps 7 and 8 are  
 84075           skipped or processed.
- 84076           7. If there are any trailing <slash> characters in *string*, they shall be removed.
- 84077           8. If the remaining *string* is empty, *string* shall be set to a single <slash> character.

84078           The resulting string shall be written to standard output.

84079 **OPTIONS**

84080           None.

84081 **OPERANDS**

84082           The following operand shall be supported:

84083           *string*           A string.84084 **STDIN**

84085           Not used.

84086 **INPUT FILES**

84087           None.

84088 **ENVIRONMENT VARIABLES**84089           The following environment variables shall affect the execution of *dirname*:

- |       |                 |                                                                                                                                                                                                                                          |
|-------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 84090 | <i>LANG</i>     | Provide a default value for the internationalization variables that are unset or null.<br>(See XBD Section 8.2 (on page 174) for the precedence of internationalization<br>variables used to determine the values of locale categories.) |
| 84093 | <i>LC_ALL</i>   | If set to a non-empty string value, override the values of all the other<br>internationalization variables.                                                                                                                              |
| 84095 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as<br>characters (for example, single-byte as opposed to multi-byte characters in<br>arguments).                                                          |

84098 *LC\_MESSAGES*  
 84099 Determine the locale that should be used to affect the format and contents of  
 84100 diagnostic messages written to standard error.

84101 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 84102 ASYNCHRONOUS EVENTS

84103 Default.

#### 84104 STDOUT

84105 The *dirname* utility shall write a line to the standard output in the following format:

84106 "%s\n", <resulting string>

#### 84107 STDERR

84108 The standard error shall be used only for diagnostic messages.

#### 84109 OUTPUT FILES

84110 None.

#### 84111 EXTENDED DESCRIPTION

84112 None.

#### 84113 EXIT STATUS

84114 The following exit values shall be returned:

84115 0 Successful completion.

84116 >0 An error occurred.

#### 84117 CONSEQUENCES OF ERRORS

84118 Default.

#### 84119 APPLICATION USAGE

84120 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 84121 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters  
 84122 to the beginning of a *pathname* unless they can ensure that there are more or less than two or are  
 84123 prepared to deal with the implementation-defined consequences.

#### 84124 EXAMPLES

84125

84126

84127

84128

84129

84130

84131

84132

84133

84134

84135

| Command                 | Results     |
|-------------------------|-------------|
| <i>dirname</i> /        | /           |
| <i>dirname</i> //       | / or //     |
| <i>dirname</i> /a/b/    | /a          |
| <i>dirname</i> //a//b// | //a         |
| <i>dirname</i>          | Unspecified |
| <i>dirname</i> a        | .( \$? = 0) |
| <i>dirname</i> ""       | .( \$? = 0) |
| <i>dirname</i> /a       | /           |
| <i>dirname</i> /a/b     | /a          |
| <i>dirname</i> a/b      | a           |

#### 84136 RATIONALE

84137 The *dirname* utility originated in System III. It has evolved through the System V releases to a  
 84138 version that matches the requirements specified in this description in System V Release 3. 4.3  
 84139 BSD and earlier versions did not include *dirname*.

84140 The behaviors of *basename* and *dirname* in this volume of POSIX.1-2008 have been coordinated so

**dirname**

Utilities

84141 that when *string* is a valid pathname:

84142 `$(basename "string")`

84143 would be a valid filename for the file in the directory:

84144 `$(dirname "string")`

84145 This would not work for the versions of these utilities in early proposals due to the way  
84146 processing of trailing <slash> characters was specified. Consideration was given to leaving  
84147 processing unspecified if there were trailing <slash> characters, but this cannot be done. XBD  
84148 [Section 3.266](#) (on page 75) allows trailing <slash> characters. The *basename* and *dirname* utilities  
84149 have to specify consistent handling for all valid pathnames.

84150 **FUTURE DIRECTIONS**

84151 None.

84152 **SEE ALSO**

84153 [Section 2.5](#) (on page 2301), *basename*

84154 XBD [Section 3.266](#) (on page 75), [Chapter 8](#) (on page 173)

84155 **CHANGE HISTORY**

84156 First released in Issue 2.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

84157 **NAME**

84158 du — estimate file space usage

84159 **SYNOPSIS**84160 du [-a|-s] [-kx] [-H|-L] [*file*...]84161 **DESCRIPTION**

84162 By default, the *du* utility shall write to standard output the size of the file space allocated to, and  
 84163 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the  
 84164 specified files. By default, when a symbolic link is encountered on the command line or in the  
 84165 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the  
 84166 link), and shall not follow the link to another portion of the file hierarchy. The size of the file  
 84167 space allocated to a file of type directory shall be defined as the sum total of space allocated to  
 84168 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

84169 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the  
 84170 final exit status is affected. Files with multiple links shall be counted and written for only one  
 84171 entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall  
 84172 be written in 512-byte units, rounded up to the next 512-byte unit.

84173 **OPTIONS**84174 The *du* utility shall conform to XBD Section 12.2 (on page 215).

84175 The following options shall be supported:

84176 **-a** In addition to the default output, report the size of each file not of type directory in  
 84177 the file hierarchy rooted in the specified file. Regardless of the presence of the **-a**  
 84178 option, non-directories given as *file* operands shall always be listed.

84179 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the  
 84180 file or file hierarchy referenced by the link.

84181 **-k** Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.

84182 **-L** If a symbolic link is specified on the command line or encountered during the  
 84183 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy  
 84184 referenced by the link.

84185 **-s** Instead of the default output, report only the total sum for each of the specified  
 84186 files.

84187 **-x** When evaluating file sizes, evaluate only those files that have the same device as  
 84188 the file specified by the *file* operand.

84189 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 84190 an error. The last option specified shall determine the behavior of the utility.

84191 **OPERANDS**

84192 The following operand shall be supported:

84193 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current  
 84194 directory shall be used.

84195 **STDIN**

84196 Not used.

84197 **INPUT FILES**

84198 None.

84199 **ENVIRONMENT VARIABLES**84200 The following environment variables shall affect the execution of *du*:84201 *LANG* Provide a default value for the internationalization variables that are unset or null.  
84202 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
84203 variables used to determine the values of locale categories.)84204 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
84205 internationalization variables.84206 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
84207 characters (for example, single-byte as opposed to multi-byte characters in  
84208 arguments).84209 *LC\_MESSAGES*84210 Determine the locale that should be used to affect the format and contents of  
84211 diagnostic messages written to standard error.84212 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.84213 **ASYNCHRONOUS EVENTS**

84214 Default.

84215 **STDOUT**84216 The output from *du* shall consist of the amount of space allocated to a file and the name of the  
84217 file, in the following format:

84218 "%d %s\n", &lt;size&gt;, &lt;pathname&gt;

84219 **STDERR**

84220 The standard error shall be used only for diagnostic messages.

84221 **OUTPUT FILES**

84222 None.

84223 **EXTENDED DESCRIPTION**

84224 None.

84225 **EXIT STATUS**

84226 The following exit values shall be returned:

84227 0 Successful completion.

84228 &gt;0 An error occurred.

84229 **CONSEQUENCES OF ERRORS**

84230 Default.

84231 **APPLICATION USAGE**

84232 None.

84233 **EXAMPLES**

84234 None.

84235 **RATIONALE**84236 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other  
84237 utilities in this volume of POSIX.1-2008. This does not mandate that the file system itself be  
84238 based on 512-byte blocks. The *-k* option was added as a compromise measure. It was agreed by  
84239 the standard developers that 512 bytes was the best default unit because of its complete  
84240 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
84241 that a *-k* option to switch to 1024-byte units was a good compromise. Users who prefer the

84242 1 024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts  
84243 relying on the 512-byte units.

84244 The **-b** option was added to an early proposal to provide a resolution to the situation where  
84245 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-  
84246 defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for  
84247 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as  
84248 512-byte units.

84249 Historical file systems provided no way to obtain exact figures for the space allocation given to  
84250 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*  
84251 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is  
84252 space used by the file system in the storage of the file, but that need not be counted in the space  
84253 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position  
84254 beyond the end of the file and data has subsequently been written at that point. A file system  
84255 need not allocate all the intervening zero-filled blocks to such a file. It is up to the  
84256 implementation to define exactly how accurate its methods are.

84257 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell  
84258 and Utilities description is implied by the language in the SVID where **-s** is described as causing  
84259 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly  
84260 Conforming POSIX Shell and Utilities Application cannot use that combination.

84261 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not  
84262 listing non-directories explicitly given as operands, unless the **-a** option is specified, was  
84263 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default  
84264 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no  
84265 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and  
84266 Utilities default behavior shall be to produce such messages. These messages can be turned off  
84267 with shell redirection to achieve the System V behavior.

84268 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of  
84269 POSIX.1-2008 because there was no other historical method of limiting the *du* search to a single  
84270 file hierarchy. This limitation of the search is necessary to make it possible to obtain file space  
84271 usage information about a file system on which other file systems are mounted, without having  
84272 to resort to a lengthy *find* and *awk* script.

#### 84273 FUTURE DIRECTIONS

84274 None.

#### 84275 SEE ALSO

84276 *ls*

84277 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

84278 XSH *fstatat()*

#### 84279 CHANGE HISTORY

84280 First released in Issue 2.

#### 84281 Issue 6

84282 This utility is marked as part of the User Portability Utilities option.

84283 The APPLICATION USAGE section is added.

84284 The obsolescent **-r** option is removed.

84285 The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had  
84286 incorrectly been marked LEGACY in Issue 5.

**du***Utilities*

- 84287 The **-H** and **-L** options for symbolic links are added as described in the IEEE P1003.2b draft standard.
- 84288
- 84289 **Issue 7**
- 84290 The *du* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.
- 84291
- 84292 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

84293 **NAME**

84294 echo — write arguments to standard output

84295 **SYNOPSIS**84296 echo [*string...*]84297 **DESCRIPTION**84298 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are  
84299 no arguments, only the <newline> is written.84300 **OPTIONS**84301 The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10  
84302 of XBD Section 12.2 (on page 215); "--" shall be recognized as a string operand.

84303 Implementations shall not support any options.

84304 **OPERANDS**

84305 The following operands shall be supported:

84306 *string* A string to be written to standard output. If the first operand is **-n**, or if any of the  
84307 operands contain a <backslash> character, the results are implementation-defined.84308 XSI On XSI-conformant systems, if the first operand is **-n**, it shall be treated as a string,  
84309 not an option. The following character sequences shall be recognized on XSI-  
84310 conformant systems within any of the arguments:

84311 \a Write an &lt;alert&gt;.

84312 \b Write a &lt;backspace&gt;.

84313 \c Suppress the <newline> that otherwise follows the final argument in the  
84314 output. All characters following the '\c' in the arguments shall be  
84315 ignored.

84316 \f Write a &lt;form-feed&gt;.

84317 \n Write a &lt;newline&gt;.

84318 \r Write a &lt;carriage-return&gt;.

84319 \t Write a &lt;tab&gt;.

84320 \v Write a &lt;vertical-tab&gt;.

84321 \ Write a &lt;backslash&gt; character.

84322 \0*num* Write an 8-bit value that is the zero, one, two, or three-digit octal number  
84323 *num*.84324 **STDIN**

84325 Not used.

84326 **INPUT FILES**

84327 None.

84328 **ENVIRONMENT VARIABLES**84329 The following environment variables shall affect the execution of *echo*:84330 *LANG* Provide a default value for the internationalization variables that are unset or null.  
84331 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
84332 variables used to determine the values of locale categories.)

## echo

Utilities

|       |     |                               |                                                                                                                                                                                                                                                      |
|-------|-----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 84333 |     | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                             |
| 84334 |     |                               |                                                                                                                                                                                                                                                      |
| 84335 | XSI | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                            |
| 84336 |     |                               |                                                                                                                                                                                                                                                      |
| 84337 |     |                               |                                                                                                                                                                                                                                                      |
| 84338 |     | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                      |
| 84339 |     |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                         |
| 84340 |     |                               |                                                                                                                                                                                                                                                      |
| 84341 | XSI | <i>NLSPATH</i>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                |
| 84342 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                      |
| 84343 |     |                               | Default.                                                                                                                                                                                                                                             |
| 84344 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                      |
| 84345 |     |                               | The <i>echo</i> utility arguments shall be separated by single <space> characters and a <newline> character shall follow the last argument. Output transformations shall occur based on the escape sequences in the input. See the OPERANDS section. |
| 84346 | XSI |                               |                                                                                                                                                                                                                                                      |
| 84347 |     |                               |                                                                                                                                                                                                                                                      |
| 84348 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                      |
| 84349 |     |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                       |
| 84350 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                      |
| 84351 |     |                               | None.                                                                                                                                                                                                                                                |
| 84352 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                      |
| 84353 |     |                               | None.                                                                                                                                                                                                                                                |
| 84354 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                      |
| 84355 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                         |
| 84356 |     | 0                             | Successful completion.                                                                                                                                                                                                                               |
| 84357 |     | >0                            | An error occurred.                                                                                                                                                                                                                                   |
| 84358 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                      |
| 84359 |     |                               | Default.                                                                                                                                                                                                                                             |
| 84360 |     | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                      |
| 84361 |     |                               | It is not possible to use <i>echo</i> portably across all POSIX systems unless both <i>-n</i> (as the first argument) and escape sequences are omitted.                                                                                              |
| 84362 |     |                               |                                                                                                                                                                                                                                                      |
| 84363 |     |                               | The <i>printf</i> utility can be used portably to emulate any of the traditional behaviors of the <i>echo</i> utility as follows (assuming that <i>IFS</i> has its standard value or is unset):                                                      |
| 84364 |     |                               |                                                                                                                                                                                                                                                      |
| 84365 |     |                               | • The historic System V <i>echo</i> and the requirements on XSI implementations in this volume of POSIX.1-2008 are equivalent to:                                                                                                                    |
| 84366 |     |                               |                                                                                                                                                                                                                                                      |
| 84367 |     |                               | <pre>printf "%b\n" "\$*" </pre>                                                                                                                                                                                                                      |
| 84368 |     |                               | • The BSD <i>echo</i> is equivalent to:                                                                                                                                                                                                              |
| 84369 |     |                               | <pre>if [ "X\$1" = "X-n" ] then     shift     printf "%s" "\$*" else     printf "%s\n" "\$*" fi </pre>                                                                                                                                               |
| 84370 |     |                               |                                                                                                                                                                                                                                                      |
| 84371 |     |                               |                                                                                                                                                                                                                                                      |
| 84372 |     |                               |                                                                                                                                                                                                                                                      |
| 84373 |     |                               |                                                                                                                                                                                                                                                      |
| 84374 |     |                               |                                                                                                                                                                                                                                                      |
| 84375 |     |                               |                                                                                                                                                                                                                                                      |

84376 New applications are encouraged to use *printf* instead of *echo*.

#### 84377 EXAMPLES

84378 None.

#### 84379 RATIONALE

84380 The *echo* utility has not been made obsolescent because of its extremely widespread use in  
84381 historical applications. Conforming applications that wish to do prompting without <newline>  
84382 characters or that could possibly be expecting to echo a **-n**, should use the *printf* utility derived  
84383 from the Ninth Edition system.

84384 As specified, *echo* writes its arguments in the simplest of ways. The two different historical  
84385 versions of *echo* vary in fatally incompatible ways.

84386 The BSD *echo* checks the first argument for the string **-n** which causes it to suppress the  
84387 <newline> that would otherwise follow the final argument in the output.

84388 The System V *echo* does not support any options, but allows escape sequences within its  
84389 operands, as described for XSI implementations in the OPERANDS section.

84390 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications  
84391 depend on *echo* to echo *all* of its arguments, except for the **-n** option in the BSD version.

#### 84392 FUTURE DIRECTIONS

84393 None.

#### 84394 SEE ALSO

84395 *printf*

84396 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

#### 84397 CHANGE HISTORY

84398 First released in Issue 2.

#### 84399 Issue 5

84400 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”  
84401 support any options; in the previous issue this said “need not”.

#### 84402 Issue 6

84403 The following new requirements on POSIX implementations derive from alignment with the  
84404 Single UNIX Specification:

- 84405 • A set of character sequences is defined as *string* operands.
- 84406 • *LC\_CTYPE* is added to the list of environment variables affecting *echo*.
- 84407 • In the OPTIONS section, implementations shall not support any options.

84408 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can  
84409 accommodate historical BSD behavior.

#### 84410 Issue 7

84411 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**ed**84412 **NAME**

84413 ed — edit text

84414 **SYNOPSIS**84415 ed [-p *string*] [-s] [*file*]84416 **DESCRIPTION**

84417 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In  
 84418 command mode the input characters shall be interpreted as commands, and in input mode they  
 84419 shall be interpreted as text. See the EXTENDED DESCRIPTION section.

84420 If an operand is '-', the results are unspecified.

84421 **OPTIONS**

84422 The *ed* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage  
 84423 of '-'.

84424 The following options shall be supported:

84425 **-p** *string* Use *string* as the prompt string when in command mode. By default, there shall be  
 84426 no prompt string.

84427 **-s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'  
 84428 prompt after a *!command*.

84429 **OPERANDS**

84430 The following operand shall be supported:

84431 *file* If the *file* argument is given, *ed* shall simulate an **e** command on the file named by  
 84432 the pathname, *file*, before accepting commands from the standard input.

84433 **STDIN**

84434 The standard input shall be a text file consisting of commands, as described in the EXTENDED  
 84435 DESCRIPTION section.

84436 **INPUT FILES**

84437 The input files shall be text files.

84438 **ENVIRONMENT VARIABLES**84439 The following environment variables shall affect the execution of *ed*:

84440 **HOME** Determine the pathname of the user's home directory.

84441 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 84442 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 84443 variables used to determine the values of locale categories.)

84444 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 84445 internationalization variables.

84446 **LC\_COLLATE**

84447 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 84448 character collating elements within regular expressions.

84449 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 84450 characters (for example, single-byte as opposed to multi-byte characters in  
 84451 arguments and input files) and the behavior of character classes within regular  
 84452 expressions.

- 84453 *LC\_MESSAGES*
- 84454 Determine the locale that should be used to affect the format and contents of
- 84455 diagnostic messages written to standard error and informative messages written to
- 84456 standard output.
- 84457 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 84458 **ASYNCHRONOUS EVENTS**
- 84459 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
- 84460 section in [Section 1.4](#), on page 2288) with the following exceptions:
- 84461 **SIGINT** The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
- 84462 output, and return to command mode (see the EXTENDED DESCRIPTION
- 84463 section).
- 84464 **SIGHUP** If the buffer is not empty and has changed since the last write, the *ed* utility shall
- 84465 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the
- 84466 current directory shall be used; if that fails, the file named **ed.hup** in the directory
- 84467 named by the *HOME* environment variable shall be used. In any case, the *ed* utility
- 84468 shall exit without writing the file to the currently remembered pathname and
- 84469 without returning to command mode.
- 84470 **SIGQUIT** The *ed* utility shall ignore this event.
- 84471 **STDOUT**
- 84472 Various editing commands and the prompting feature (see **-p**) write to standard output, as
- 84473 described in the EXTENDED DESCRIPTION section.
- 84474 **STDERR**
- 84475 The standard error shall be used only for diagnostic messages.
- 84476 **OUTPUT FILES**
- 84477 The output files shall be text files whose formats are dependent on the editing commands given.
- 84478 **EXTENDED DESCRIPTION**
- 84479 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
- 84480 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
- 84481 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a
- 84482 single-character *command*, possibly followed by parameters to that command. These addresses
- 84483 specify one or more lines in the buffer. Every command that requires addresses has default
- 84484 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the
- 84485 prompt string shall be written to standard output before each command is read.
- 84486 In general, only one command can appear on a line. Certain commands allow text to be input.
- 84487 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be
- 84488 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.
- 84489 Input mode is terminated by entering a line consisting of two characters: a <period> ( '.' )
- 84490 followed by a <newline>. This line is not considered part of the input text.

84491 **Regular Expressions in ed**

84492 The *ed* utility shall support basic regular expressions, as described in XBD Section 9.3 (on page  
84493 183). Since regular expressions in *ed* are always matched against single lines (excluding the  
84494 terminating <newline> characters), never against any larger section of text, there is no way for a  
84495 regular expression to match a <newline>.

84496 A null RE shall be equivalent to the last RE encountered.

84497 Regular expressions are used in addresses to specify lines, and in some commands (for example,  
84498 the *s* substitute command) to specify portions of a line to be substituted.

84499 **Addresses in ed**

84500 Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a  
84501 command. The current line number is the address of the current line. If the edit buffer is not  
84502 empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

84503 Addresses shall be constructed as follows:

- 84504 1. The <period> character ( ' . ' ) shall address the current line.
- 84505 2. The <dollar-sign> character ( ' \$ ' ) shall address the last line of the edit buffer.
- 84506 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 84507 4. The <apostrophe>-*x* character pair ( " ' x " ) shall address the line marked with the mark  
84508 name character *x*, which shall be a lowercase letter from the portable character set. It shall  
84509 be an error if the character has not been set to mark a line or if the line that was marked is  
84510 not currently present in the edit buffer.
- 84511 5. A BRE enclosed by <slash> characters ( ' / ' ) shall address the first line found by  
84512 searching forwards from the line following the current line toward the end of the edit  
84513 buffer and stopping at the first line for which the line excluding the terminating  
84514 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of  
84515 <slash> characters shall address the next line for which the line excluding the terminating  
84516 <newline> matches the last BRE encountered. In addition, the second <slash> can be  
84517 omitted at the end of a command line. Within the BRE, a <backslash>-<slash> pair ( " \ / " )  
84518 shall represent a literal <slash> instead of the BRE delimiter. If necessary, the search shall  
84519 wrap around to the beginning of the buffer and continue up to and including the current  
84520 line, so that the entire buffer is searched.
- 84521 6. A BRE enclosed by <question-mark> characters ( ' ? ' ) shall address the first line found by  
84522 searching backwards from the line preceding the current line toward the beginning of the  
84523 edit buffer and stopping at the first line for which the line excluding the terminating  
84524 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of  
84525 <question-mark> characters ( " ? ? " ) shall address the previous line for which the line  
84526 excluding the terminating <newline> matches the last BRE encountered. In addition, the  
84527 second <question-mark> can be omitted at the end of a command line. Within the BRE, a  
84528 <backslash>-<question-mark> pair ( " \ ? " ) shall represent a literal <question-mark>  
84529 instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the  
84530 buffer and continue up to and including the current line, so that the entire buffer is  
84531 searched.
- 84532 7. A <plus-sign> ( ' + ' ) or <hyphen> character ( ' - ' ) followed by a decimal number shall  
84533 address the current line plus or minus the number. A <plus-sign> or <hyphen> character  
84534 not followed by a decimal number shall address the current line plus or minus 1.

84535 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
84536 Address offsets are constructed as follows:

- 84537 • A <plus-sign> or <hyphen> character followed by a decimal number shall add or subtract,  
84538 respectively, the indicated number of lines to or from the address. A <plus-sign> or  
84539 <hyphen> character not followed by a decimal number shall add or subtract 1 to or from  
84540 the address.
- 84541 • A decimal number shall add the indicated number of lines to the address.

84542 It shall not be an error for an intermediate address value to be less than zero or greater than the  
84543 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
84544 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a  
84545 matching line.

84546 Commands accept zero, one, or two addresses. If more than the required number of addresses  
84547 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more  
84548 than the required number of addresses are provided to a command, the addresses specified first  
84549 shall be evaluated and then discarded until the maximum number of valid addresses remain, for  
84550 the specified command.

84551 Addresses shall be separated from each other by a <comma> (','') or <semicolon> character  
84552 (';'). In the case of a <semicolon> separator, the current line ('.') shall be set to the first  
84553 address, and only then will the second address be calculated. This feature can be used to  
84554 determine the starting line for forwards and backwards searches; see rules 5. and 6.

84555 Addresses can be omitted on either side of the <comma> or <semicolon> separator, in which  
84556 case the resulting address pairs shall be as follows:

| Specified | Resulting   |
|-----------|-------------|
| ,         | 1 , \$      |
| , addr    | 1 , addr    |
| addr ,    | addr , addr |
| ;         | . ; \$      |
| ; addr    | . ; addr    |
| addr ;    | addr ; addr |

84564 Any <blank> characters included between addresses, address separators, or address offsets shall  
84565 be ignored.

#### 84566 Commands in ed

84567 In the following list of *ed* commands, the default addresses are shown in parentheses. The  
84568 number of addresses shown in the default shall be the number expected by the command. The  
84569 parentheses are not part of the address; they show that the given addresses are the default.

84570 It is generally invalid for more than one command to appear on a line. However, any command  
84571 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for  
84572 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be  
84573 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used  
84574 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but  
84575 it is unspecified whether the suffix writes the current line again in the requested format or  
84576 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**  
84577 suffix) shall either write just the current line or write it twice—once as specified for **p** and once  
84578 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

84579 Each address component can be preceded by zero or more <blank> characters. The command

- 84580 letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given,  
84581 the application shall ensure that it immediately follows the command.
- 84582 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the  
84583 command letter by one or more <blank> characters.
- 84584 If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed*  
84585 shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.  
84586 The *ed* utility shall write the string:
- 84587 " ? \n "
- 84588 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
84589 standard output and shall continue in command mode with the current line number unchanged.  
84590 If the **e** or **q** command is repeated with no intervening command, it shall take effect.
- 84591 If a terminal disconnect (see XBD Chapter 11 (on page 199), Modem Disconnect and Closing a  
84592 Device Terminal), is detected:
- 84593 • If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the  
84594 ASYNCHRONOUS EVENTS section for a SIGHUP signal.
  - 84595 • If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been  
84596 detected on standard input.
- 84597 If an end-of-file is detected on standard input:
- 84598 • If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command  
84599 mode. It is unspecified if any partially entered lines (that is, input text without a  
84600 terminating <newline>) are discarded from the input text.
  - 84601 • If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.
- 84602 If the closing delimiter of an RE or of a replacement string (for example, ' / ' ) in a **g**, **G**, **s**, **v**, or **V**  
84603 command would be the last character before a <newline>, that delimiter can be omitted, in  
84604 which case the addressed line shall be written. For example, the following pairs of commands  
84605 are equivalent:
- 84606 *s/s1/s2*      *s/s1/s2/p*  
84607 *g/s1*          *g/s1/p*  
84608 *?s1*            *?s1?*
- 84609 If an invalid command is entered, *ed* shall write the string:
- 84610 " ? \n "
- 84611 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
84612 standard output and shall continue in command mode with the current line number unchanged.
- 84613 **Append Command**
- 84614 *Synopsis:*      (.) *a*  
84615                    <*text*>  
84616                    .
- 84617 The **a** command shall read the given text and append it after the addressed line; the current line  
84618 number shall become the address of the last inserted line or, if there were none, the addressed  
84619 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at  
84620 the beginning of the buffer.

84621 **Change Command**

84622 *Synopsis:*     (.,.)c  
 84623                 <text>  
 84624                 .

84625         The **c** command shall delete the addressed lines, then accept input text that replaces these lines;  
 84626         the current line shall be set to the address of the last line input; or, if there were none, at the line  
 84627         after the last line deleted; if the lines deleted were originally at the end of the buffer, the current  
 84628         line number shall be set to the address of the new last line; if no lines remain in the buffer, the  
 84629         current line number shall be set to zero. Address 0 shall be valid for this command; it shall be  
 84630         interpreted as if address 1 were specified.

84631 **Delete Command**

84632 *Synopsis:*     (.,.)d

84633         The **d** command shall delete the addressed lines from the buffer. The address of the line after the  
 84634         last line deleted shall become the current line number; if the lines deleted were originally at the  
 84635         end of the buffer, the current line number shall be set to the address of the new last line; if no  
 84636         lines remain in the buffer, the current line number shall be set to zero.

84637 **Edit Command**

84638 *Synopsis:*     e [*file*]

84639         The **e** command shall delete the entire contents of the buffer and then read in the file named by  
 84640         the pathname *file*. The current line number shall be set to the address of the last line of the  
 84641         buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see  
 84642         the **f** command). The number of bytes read shall be written to standard output, unless the **-s**  
 84643         option was specified, in the following format:

84644         "%d\n", <number of bytes read>

84645         The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,  
 84646         and **w** commands. If *file* is replaced by '!', the rest of the line shall be taken to be a shell  
 84647         command line whose output is to be read. Such a shell command line shall not be remembered  
 84648         as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.  
 84649         If the buffer has changed since the last time the entire buffer was written, the user shall be  
 84650         warned, as described previously.

84651 **Edit Without Checking Command**

84652 *Synopsis:*     E [*file*]

84653         The **E** command shall possess all properties and restrictions of the **e** command except that the  
 84654         editor shall not check to see whether any changes have been made to the buffer since the last **w**  
 84655         command.

84656 **Filename Command**84657 *Synopsis:*    *f* [*file*]

84658 If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether  
 84659 the name is changed or not, it shall then write the (possibly new) currently remembered  
 84660 pathname to the standard output in the following format:

84661 "%s\n", &lt;pathname&gt;

84662 The current line number shall be unchanged.

84663 **Global Command**84664 *Synopsis:*    (1, \$) *g*/RE/*command list*

84665 In the **g** command, the first step shall be to mark every line for which the line excluding the  
 84666 terminating <newline> matches the given RE. Then, going sequentially from the beginning of  
 84667 the file to the end of the file, the given *command list* shall be executed for each marked line, with  
 84668 the current line number set to the address of that line. Any line modified by the *command list*  
 84669 shall be unmarked. When the **g** command completes, the current line number shall have the  
 84670 value assigned by the last command in the *command list*. If there were no matching lines, the  
 84671 current line number shall not be changed. A single command or the first of a list of commands  
 84672 shall appear on the same line as the global command. All lines of a multi-line list except the last  
 84673 line shall be ended with a <backslash> preceding the terminating <newline>; the **a**, **i**, and **c**  
 84674 commands and associated input are permitted. The '.' terminating input mode can be omitted  
 84675 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p**  
 84676 command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined  
 84677 results. Any character other than <space> or <newline> can be used instead of a <slash> to  
 84678 delimit the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is  
 84679 preceded by a <backslash>.

84680 **Interactive Global Command**84681 *Synopsis:*    (1, \$) *G*/RE/

84682 In the **G** command, the first step shall be to mark every line for which the line excluding the  
 84683 terminating <newline> matches the given RE. Then, for every such line, that line shall be  
 84684 written, the current line number shall be set to the address of that line, and any one command  
 84685 (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline>  
 84686 shall act as a null command (causing no action to be taken on the current line); an '&' shall  
 84687 cause the re-execution of the most recent non-null command executed within the current  
 84688 invocation of **G**. Note that the commands input as part of the execution of the **G** command can  
 84689 address and affect any lines in the buffer. Any line modified by the command shall be  
 84690 unmarked. The final value of the current line number shall be the value set by the last command  
 84691 successfully executed. (Note that the last command successfully executed shall be the **G**  
 84692 command itself if a command fails or the null command is specified.) If there were no matching  
 84693 lines, the current line number shall not be changed. The **G** command can be terminated by a  
 84694 SIGINT signal. Any character other than <space> or <newline> can be used instead of a <slash>  
 84695 to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a  
 84696 literal character if it is preceded by a <backslash>.

84697 **Help Command**84698 *Synopsis:* h84699 The **h** command shall write a short message to standard output that explains the reason for the  
84700 most recent '?' notification. The current line number shall be unchanged.84701 **Help-Mode Command**84702 *Synopsis:* H84703 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)  
84704 shall be written to standard output for all subsequent '?' notifications. The **H** command  
84705 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,  
84706 the **H** command also explains the previous '?' notification, if there was one. The current line  
84707 number shall be unchanged.84708 **Insert Command**84709 *Synopsis:* (.) i  
84710 <text>  
84711 .84712 The **i** command shall insert the given text before the addressed line; the current line is set to the  
84713 last inserted line or, if there was none, to the addressed line. This command differs from the **a**  
84714 command only in the placement of the input text. Address 0 shall be valid for this command; it  
84715 shall be interpreted as if address 1 were specified.84716 **Join Command**84717 *Synopsis:* (.,.+1) j84718 The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If  
84719 exactly one address is given, this command shall do nothing. If lines are joined, the current line  
84720 number shall be set to the address of the joined line; otherwise, the current line number shall be  
84721 unchanged.84722 **Mark Command**84723 *Synopsis:* (.) x x84724 The **k** command shall mark the addressed line with name *x*, which the application shall ensure  
84725 is a lowercase letter from the portable character set. The address "' x'" shall then refer to this  
84726 line; the current line number shall be unchanged.84727 **List Command**84728 *Synopsis:* (.,.) l84729 The **l** command shall write to standard output the addressed lines in a visually unambiguous  
84730 form. The characters listed in XBD Table 5-1 (on page 121) ('\ ', '\a', '\b', '\f', '\r',  
84731 '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not  
84732 applicable. Non-printable characters not in the table shall be written as one three-digit octal  
84733 number (with a preceding <backslash> character) for each byte in the character (most significant  
84734 byte first).84735 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a  
84736 <backslash>; the length at which folding occurs is unspecified, but should be appropriate for the

84737 output device. The end of each line shall be marked with a '\$', and '\$' characters within the  
 84738 text shall be written with a preceding <backslash>. An I command can be appended to any  
 84739 other command other than e, E, f, q, Q, r, w, or !. The current line number shall be set to the  
 84740 address of the last line written.

#### 84741 **Move Command**

84742 *Synopsis:* (.,.)*address*

84743 The m command shall reposition the addressed lines after the line addressed by *address*.  
 84744 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning  
 84745 of the buffer. It shall be an error if *address* falls within the range of moved lines. The  
 84746 current line number shall be set to the address of the last line moved.

#### 84747 **Number Command**

84748 *Synopsis:* (.,.)*n*

84749 The n command shall write to standard output the addressed lines, preceding each line by its  
 84750 line number and a <tab>; the current line number shall be set to the address of the last line  
 84751 written. The n command can be appended to any command other than e, E, f, q, Q, r, w, or !.

#### 84752 **Print Command**

84753 *Synopsis:* (.,.)*p*

84754 The p command shall write to standard output the addressed lines; the current line number shall  
 84755 be set to the address of the last line written. The p command can be appended to any command  
 84756 other than e, E, f, q, Q, r, w, or !.

#### 84757 **Prompt Command**

84758 *Synopsis:* P

84759 The P command shall cause *ed* to prompt with an <asterisk> ('\*') (or *string*, if -p is specified)  
 84760 for all subsequent commands. The P command alternatively shall turn this mode on and off; it  
 84761 shall be initially on if the -p option is specified; otherwise, off. The current line number shall be  
 84762 unchanged.

#### 84763 **Quit Command**

84764 *Synopsis:* q

84765 The q command shall cause *ed* to exit. If the buffer has changed since the last time the entire  
 84766 buffer was written, the user shall be warned, as described previously.

#### 84767 **Quit Without Checking Command**

84768 *Synopsis:* Q

84769 The Q command shall cause *ed* to exit without checking whether changes have been made in the  
 84770 buffer since the last w command.

84771 **Read Command**84772 *Synopsis:* (\$)r [*file*]

84773 The **r** command shall read in the file named by the pathname *file* and append it after the  
 84774 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be  
 84775 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed  
 84776 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file  
 84777 to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the  
 84778 number of bytes read shall be written to standard output in the following format:

84779 "%d\n", &lt;number of bytes read&gt;

84780 The current line number shall be set to the address of the last line read in. If *file* is replaced by  
 84781 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.  
 84782 Such a shell command line shall not be remembered as the current pathname.

84783 **Substitute Command**84784 *Synopsis:* (.,.)s/RE/replacement/flags

84785 The **s** command shall search each addressed line for an occurrence of the specified RE and  
 84786 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the  
 84787 following description of the **g** suffix. It is an error if the substitution fails on every addressed  
 84788 line. Any character other than <space> or <newline> can be used instead of a <slash> to delimit  
 84789 the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal  
 84790 character if it is preceded by a <backslash>. The current line shall be set to the address of the  
 84791 last line on which a substitution occurred.

84792 An <ampersand> ('&') appearing in the *replacement* shall be replaced by the string matching the  
 84793 RE on the current line. The special meaning of '&' in this context can be suppressed by  
 84794 preceding it by <backslash>. As a more general feature, the characters '\n', where *n* is a digit,  
 84795 shall be replaced by the text matched by the corresponding back-reference expression. If the  
 84796 corresponding back-reference expression does not match, then the characters '\n' shall be  
 84797 replaced by the empty string. When the character '%' is the only character in the *replacement*, the  
 84798 *replacement* used in the most recent substitute command shall be used as the *replacement* in the  
 84799 current substitute command; if there was no previous substitute command, the use of '%' in this  
 84800 manner shall be an error. The '%' shall lose its special meaning when it is in a replacement  
 84801 string of more than one character or is preceded by a <backslash>. For each <backslash>  
 84802 encountered in scanning *replacement* from beginning to end, the following character shall lose its  
 84803 special meaning (if any). It is unspecified what special meaning is given to any character other  
 84804 than <backslash>, '&', '%', or digits.

84805 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the  
 84806 <newline> in the *replacement* by preceding it by <backslash>. Such substitution cannot be done  
 84807 as part of a **g** or **v** command list. The current line number shall be set to the address of the last  
 84808 line on which a substitution is performed. If no substitution is performed, the current line  
 84809 number shall be unchanged. If a line is split, a substitution shall be considered to have been  
 84810 performed on each of the new lines for the purpose of determining the new current line number.  
 84811 A substitution shall be considered to have been performed even if the replacement string is  
 84812 identical to the string that it replaces.

84813 The application shall ensure that the value of *flags* is zero or more of:84814 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.

- 84815 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first  
84816 one. If both **g** and *count* are specified, the results are unspecified.
- 84817 **l** Write to standard output the final line in which a substitution was made. The line shall  
84818 be written in the format specified for the **l** command.
- 84819 **n** Write to standard output the final line in which a substitution was made. The line shall  
84820 be written in the format specified for the **n** command.
- 84821 **p** Write to standard output the final line in which a substitution was made. The line shall  
84822 be written in the format specified for the **p** command.

### 84823 Copy Command

84824 *Synopsis:* (*. , .*)*t**address*

84825 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines  
84826 shall be placed after address *address* (which can be 0); the current line number shall be set to the  
84827 address of the last line added.

### 84828 Undo Command

84829 *Synopsis:* *u*

84830 The **u** command shall nullify the effect of the most recent command that modified anything in  
84831 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes  
84832 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no  
84833 changes were made by the global command (such as with **g**/RE/**p**), the **u** command shall have  
84834 no effect. The current line number shall be set to the value it had immediately before the  
84835 command being undone started.

### 84836 Global Non-Matched Command

84837 *Synopsis:* (*1, \$*)*v*/RE/*command list*

84838 This command shall be equivalent to the global command **g** except that the lines that are marked  
84839 during the first step shall be those for which the line excluding the terminating <newline> does  
84840 not match the RE.

### 84841 Interactive Global Not-Matched Command

84842 *Synopsis:* (*1, \$*)*V*/RE/

84843 This command shall be equivalent to the interactive global command **G** except that the lines that  
84844 are marked during the first step shall be those for which the line excluding the terminating  
84845 <newline> does not match the RE.

### 84846 Write Command

84847 *Synopsis:* (*1, \$*)*w* [*file*]

84848 The **w** command shall write the addressed lines into the file named by the pathname *file*. The  
84849 command shall create the file, if it does not exist, or shall replace the contents of the existing file.  
84850 The currently remembered pathname shall not be changed unless there is no remembered  
84851 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used  
84852 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is  
84853 successful, the number of bytes written shall be written to standard output, unless the **-s** option  
84854 was specified, in the following format:

84855 "%d\n", <number of bytes written>

84856 If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose  
 84857 standard input shall be the addressed lines. Such a shell command line shall not be remembered  
 84858 as the current pathname. This usage of the write command with '!' shall not be considered as a  
 84859 "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall  
 84860 not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or  
 84861 **q** commands.

#### 84862 **Line Number Command**

84863 *Synopsis:* (\$) =

84864 The line number of the addressed line shall be written to standard output in the following  
 84865 format:

84866 "%d\n", <line number>

84867 The current line number shall be unchanged by this command.

#### 84868 **Shell Escape Command**

84869 *Synopsis:* !*command*

84870 The remainder of the line after the '!' shall be sent to the command interpreter to be  
 84871 interpreted as a shell command line. Within the text of that shell command line, the unescaped  
 84872 character '%' shall be replaced with the remembered pathname; if a '!' appears as the first  
 84873 character of the command, it shall be replaced with the text of the previous shell command  
 84874 executed via '!'. Thus, "!!" shall repeat the previous !*command*. If any replacements of '%' or  
 84875 '!' are performed, the modified line shall be written to the standard output before *command* is  
 84876 executed. The ! command shall write:

84877 "! \n"

84878 to standard output upon completion, unless the **-s** option is specified. The current line number  
 84879 shall be unchanged.

#### 84880 **Null Command**

84881 *Synopsis:* (.+1)

84882 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall  
 84883 be equivalent to "+1p". The current line number shall be set to the address of the written line.

#### 84884 **EXIT STATUS**

84885 The following exit values shall be returned:

84886 0 Successful completion without any file or command errors.

84887 >0 An error occurred.

#### 84888 **CONSEQUENCES OF ERRORS**

84889 When an error in the input script is encountered, or when an error is detected that is a  
 84890 consequence of the data (not) present in the file or due to an external condition such as a read or  
 84891 write error:

- 84892 • If the standard input is a terminal device file, all input shall be flushed, and a new  
 84893 command read.

- 84894 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

#### 84895 APPLICATION USAGE

84896 Because of the extremely terse nature of the default error messages, the prudent script writer  
84897 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some  
84898 clue as to the cause is made available.

84899 In earlier versions of this standard, an obsolescent `-` option was described. This is no longer  
84900 specified. Applications should use the `-s` option. Using `-` as a *file* operand now produces  
84901 unspecified results. This allows implementations to continue to support the former required  
84902 behavior.

#### 84903 EXAMPLES

84904 None.

#### 84905 RATIONALE

84906 The initial description of this utility was adapted from the SVID. It contains some features not  
84907 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and  
84908 BSD *ed* utilities include, but need not be limited to:

- 84909 • The BSD `-` option does not suppress the `' ! '` prompt after a `!` command.
- 84910 • BSD does not support the special meanings of the `' % '` and `' ! '` characters within a `!`  
84911 command.
- 84912 • BSD does not support the *addresses* `' ; '` and `' . '`.
- 84913 • BSD allows the command/suffix pairs `pp`, `ll`, and so on, which are unspecified in this  
84914 volume of POSIX.1-2008.
- 84915 • BSD does not support the `' ! '` character part of the `e`, `r`, or `w` commands.
- 84916 • A failed `g` command in BSD sets the line number to the last line searched if there are no  
84917 matches.
- 84918 • BSD does not default the *command list* to the `p` command.
- 84919 • BSD does not support the `G`, `h`, `H`, `n`, or `V` commands.
- 84920 • On BSD, if there is no inserted text, the insert command changes the current line to the  
84921 referenced line `-1`; that is, the line before the specified line.
- 84922 • On BSD, the *join* command with only a single address changes the current line to that  
84923 address.
- 84924 • BSD does not support the `P` command; moreover, in BSD it is synonymous with the `p`  
84925 command.
- 84926 • BSD does not support the *undo* of the commands `j`, `m`, `r`, `s`, or `t`.
- 84927 • The Version 7 *ed* command `W`, and the BSD *ed* commands `W`, `wq`, and `z` are not present in  
84928 this volume of POSIX.1-2008.

84929 The `-s` option was added to allow the functionality of the removed `-` option in a manner  
84930 compatible with the Utility Syntax Guidelines.

84931 In early proposals there was a limit, `{ED_FILE_MAX}`, that described the historical limitations of  
84932 some *ed* utilities in their handling of large files; some of these have had problems with files  
84933 larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a  
84934 *split* command in this volume of POSIX.1-2008. Since this limit was removed, this volume of  
84935 POSIX.1-2008 requires that implementations document the file size limits imposed by *ed* in the

- 84936 conformance document. The limit {ED\_LINE\_MAX} was also removed; therefore, the global  
84937 limit {LINE\_MAX} is used for input and output lines.
- 84938 The manner in which the **I** command writes non-printable characters was changed to avoid the  
84939 historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous  
84940 because most terminals simply replace overstruck characters, making the **I** format not useful for  
84941 its intended purpose of unambiguously understanding the content of the line. The historical  
84942 <backslash>-escapes were also ambiguous. (The string "a\0011" could represent a line  
84943 containing those six characters or a line containing the three characters ' a ', a byte with a binary  
84944 value of 1, and a 1.) In the format required here, a <backslash> appearing in the line is written as  
84945 "\\\" so that the output is truly unambiguous. The method of marking the ends of lines was  
84946 adopted from the *ex* editor and is required for any line ending in <space> characters; the '\$' is  
84947 placed on all lines so that a real '\$' at the end of a line cannot be misinterpreted.
- 84948 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
84949 but this has been modified in this version.
- 84950 The description of how a NUL is written was removed. The NUL character cannot be in text  
84951 files, and this volume of POSIX.1-2008 should not dictate behavior in the case of undefined,  
84952 erroneous input.
- 84953 Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands  
84954 are not patterns.
- 84955 Early proposals stated that the **-p** option worked only when standard input was associated with  
84956 a terminal device. This has been changed to conform to historical implementations, thereby  
84957 allowing applications to interpose themselves between a user and the *ed* utility.
- 84958 The form of the substitute command that uses the **n** suffix was limited in some historical  
84959 documentation (where this was described incorrectly as "backreferencing"). This limit has been  
84960 omitted because there is no reason why an editor processing lines of {LINE\_MAX} length should  
84961 have this restriction. The command **s/x/X/2047** should be able to substitute the 2 047th occurrence  
84962 of ' x ' on a line.
- 84963 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made  
84964 unspecified because BSD-based systems allow this, whereas System V does not.
- 84965 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file  
84966 have been deleted. Since this volume of POSIX.1-2008 refers to the **q** command in this instance,  
84967 such behavior is not allowed.
- 84968 Some historical implementations returned exit status zero even if command errors had occurred;  
84969 this is not allowed by this volume of POSIX.1-2008.
- 84970 Some historical implementations contained a bug that allowed a single <period> to be entered in  
84971 input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no  
84972 description of escaping any of the characters in input mode; <backslash> characters are entered  
84973 into the buffer exactly as typed. The typical method of entering a single <period> has been to  
84974 precede it with another character and then use the substitute command to delete that character.
- 84975 It is difficult under some modes of some versions of historical operating system terminal drivers  
84976 to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-2008 does not  
84977 require implementations to distinguish between the two situations, which permits historical  
84978 implementations of the *ed* utility on historical platforms to conform. Implementations are  
84979 encouraged to distinguish between the two, if possible, and take appropriate action on terminal  
84980 disconnect.
- 84981 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the

84982 start of the edit buffer. When the buffer was empty the command `.=` returned zero. POSIX.1-2008  
 84983 requires conformance to historical practice.

84984 For consistency with the `a` and `r` commands and better user functionality, the `i` and `c` commands  
 84985 must also accept an address of 0, in which case `0i` is treated as `1i` and likewise for the `c`  
 84986 command.

84987 All of the following are valid addresses:

- 84988 `+++` Three lines after the current line.
- 84989 `/pattern/-` One line before the next occurrence of pattern.
- 84990 `-2` Two lines before the current line.
- 84991 `3 ---- 2` Line one (note the intermediate negative address).
- 84992 `1 2 3` Line six.

84993 Any number of addresses can be provided to commands taking addresses; for example,  
 84994 `"1, 2, 3, 4, 5p"` prints lines 4 and 5, because two is the greatest valid number of addresses  
 84995 accepted by the `print` command. This, in combination with the `<semicolon>` delimiter, permits  
 84996 users to create commands based on ordered patterns in the file. For example, the command  
 84997 `"3; /foo/; +2p"` will display the first line after line 3 that contains the pattern `foo`, plus the next  
 84998 two lines. Note that the address `"3;"` must still be evaluated before being discarded, because  
 84999 the search origin for the `"/foo/"` command depends on this.

85000 Historically, `ed` disallowed address chains, as discussed above, consisting solely of `<comma>` or  
 85001 `<semicolon>` separators; for example, `","` or `";;"` were considered an error. For  
 85002 consistency of address specification, this restriction is removed. The following table lists some of  
 85003 the address forms now possible:

| Address  | Addr1 | Addr2 | Status     | Comment               |
|----------|-------|-------|------------|-----------------------|
| 7,       | 7     | 7     | Historical |                       |
| 7, 5,    | 5     | 5     | Historical |                       |
| 7, 5, 9  | 5     | 9     | Historical |                       |
| 7, 9     | 7     | 9     | Historical |                       |
| 7, +     | 7     | 8     | Historical |                       |
| ,        | 1     | \$    | Historical |                       |
| , 7      | 1     | 7     | Extension  |                       |
| , \$     | \$    | \$    | Extension  |                       |
| , ;      | \$    | \$    | Extension  |                       |
| 7;       | 7     | 7     | Historical |                       |
| 7; 5;    | 5     | 5     | Historical |                       |
| 7; 5; 9  | 5     | 9     | Historical |                       |
| 7; 5, 9  | 5     | 9     | Historical |                       |
| 7; \$; 4 | \$    | 4     | Historical | Valid, but erroneous. |
| 7; 9     | 7     | 9     | Historical |                       |
| 7; +     | 7     | 8     | Historical |                       |
| ;        | .     | \$    | Historical |                       |
| ; 7      | .     | 7     | Extension  |                       |
| ; ;      | \$    | \$    | Extension  |                       |
| ; ,      | \$    | \$    | Extension  |                       |

85025 Historically, `ed` accepted the `'^'` character as an address, in which case it was identical to the  
 85026 `<hyphen>` character. POSIX.1-2008 does not require or prohibit this behavior.

85027 **FUTURE DIRECTIONS**

85028 None.

85029 **SEE ALSO**85030 [Section 1.4](#) (on page 2288), *ex*, *sed*, *sh*, *vi*85031 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 215)85033 **CHANGE HISTORY**

85034 First released in Issue 2.

85035 **Issue 5**85036 In the OPTIONS section, the meaning of `-s` and `-` is clarified.

85037 A second FUTURE DIRECTION is added.

85038 **Issue 6**

85039 The obsolescent single-minus form is removed.

85040 A second APPLICATION USAGE note is added.

85041 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

85042 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

85045 The normative text is reworded to avoid use of the term “must” for application requirements.

85046 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: “Any line modified by the *command list* shall be unmarked.” to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of this standard.

85049 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

85051 **Issue 7**85052 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is `' - ' .`

85054 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

85055 SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

85057 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85058 SD5-XCU-ERN-135 is applied, removing some RATIONALE text that is no longer applicable.

85059 **NAME**

85060 env — set the environment for command invocation

85061 **SYNOPSIS**85062 env [-i] [*name=value*...]... [*utility* [*argument...*]]85063 **DESCRIPTION**85064 The *env* utility shall obtain the current environment, modify it according to its arguments, then  
85065 invoke the utility named by the *utility* operand with the modified environment.85066 Optional arguments shall be passed to *utility*.85067 If no *utility* operand is specified, the resulting environment shall be written to the standard  
85068 output, with one *name=value* pair per line.

85069 If the first argument is '-', the results are unspecified.

85070 **OPTIONS**85071 The *env* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage  
85072 of '-'.  
85073 The following options shall be supported:

85074 -i

85075 Invoke *utility* with exactly the environment specified by the arguments; the  
inherited environment shall be ignored completely.85076 **OPERANDS**

85077 The following operands shall be supported:

85078 *name=value* Arguments of the form *name=value* shall modify the execution environment, and  
85079 shall be placed into the inherited environment before the *utility* is invoked.85080 *utility* The name of the utility to be invoked. If the *utility* operand names any of the  
85081 special built-in utilities in Section 2.14 (on page 2334), the results are undefined.85082 *argument* A string to pass as an argument for the invoked utility.85083 **STDIN**

85084 Not used.

85085 **INPUT FILES**

85086 None.

85087 **ENVIRONMENT VARIABLES**85088 The following environment variables shall affect the execution of *env*:85089 *LANG* Provide a default value for the internationalization variables that are unset or null.  
85090 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
85091 variables used to determine the values of locale categories.)85092 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
85093 internationalization variables.85094 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
85095 characters (for example, single-byte as opposed to multi-byte characters in  
85096 arguments).85097 *LC\_MESSAGES*85098 Determine the locale that should be used to affect the format and contents of  
85099 diagnostic messages written to standard error.

|       |     |                               |                                                                                                                                                     |
|-------|-----|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 85100 | XSI | <b>NLSPATH</b>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                               |
| 85101 |     | <b>PATH</b>                   | Determine the location of the <i>utility</i> , as described in XBD Chapter 8 (on page 173).                                                         |
| 85102 |     |                               | If <i>PATH</i> is specified as a <i>name=value</i> operand to <i>env</i> , the <i>value</i> given shall be used in                                  |
| 85103 |     |                               | the search for <i>utility</i> .                                                                                                                     |
| 85104 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                     |
| 85105 |     |                               | Default.                                                                                                                                            |
| 85106 |     | <b>STDOUT</b>                 |                                                                                                                                                     |
| 85107 |     |                               | If no <i>utility</i> operand is specified, each <i>name=value</i> pair in the resulting environment shall be                                        |
| 85108 |     |                               | written in the form:                                                                                                                                |
| 85109 |     |                               | "%s=%s\n", <name>, <value>                                                                                                                          |
| 85110 |     |                               | If the <i>utility</i> operand is specified, the <i>env</i> utility shall not write to standard output.                                              |
| 85111 |     | <b>STDERR</b>                 |                                                                                                                                                     |
| 85112 |     |                               | The standard error shall be used only for diagnostic messages.                                                                                      |
| 85113 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                     |
| 85114 |     |                               | None.                                                                                                                                               |
| 85115 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                     |
| 85116 |     |                               | None.                                                                                                                                               |
| 85117 |     | <b>EXIT STATUS</b>            |                                                                                                                                                     |
| 85118 |     |                               | If <i>utility</i> is invoked, the exit status of <i>env</i> shall be the exit status of <i>utility</i> ; otherwise, the <i>env</i>                  |
| 85119 |     |                               | utility shall exit with one of the following values:                                                                                                |
| 85120 |     | 0                             | The <i>env</i> utility completed successfully.                                                                                                      |
| 85121 |     | 1–125                         | An error occurred in the <i>env</i> utility.                                                                                                        |
| 85122 |     | 126                           | The utility specified by <i>utility</i> was found but could not be invoked.                                                                         |
| 85123 |     | 127                           | The utility specified by <i>utility</i> could not be found.                                                                                         |
| 85124 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                     |
| 85125 |     |                               | Default.                                                                                                                                            |
| 85126 |     | <b>APPLICATION USAGE</b>      |                                                                                                                                                     |
| 85127 |     |                               | The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if |
| 85128 |     |                               | an error occurs so that applications can distinguish “failure to find a utility” from “invoked                                                      |
| 85129 |     |                               | utility exited with an error indication”. The value 127 was chosen because it is not commonly                                                       |
| 85130 |     |                               | used for other meanings; most utilities use small values for “normal error conditions” and the                                                      |
| 85131 |     |                               | values above 128 can be confused with termination due to receipt of a signal. The value 126 was                                                     |
| 85132 |     |                               | chosen in a similar manner to indicate that the utility could be found, but not invoked. Some                                                       |
| 85133 |     |                               | scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction                                                    |
| 85134 |     |                               | between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to                                                    |
| 85135 |     |                               | <i>exec</i> the utility fail with [ENOENT], and uses 126 when any attempt to <i>exec</i> the utility fails for                                      |
| 85136 |     |                               | any other reason.                                                                                                                                   |
| 85137 |     |                               | Historical implementations of the <i>env</i> utility use the <i>execop()</i> or <i>execlp()</i> functions defined in the                            |
| 85138 |     |                               | System Interfaces volume of POSIX.1-2008 to invoke the specified utility; this provides better                                                      |
| 85139 |     |                               | performance and keeps users from having to escape characters with special meaning to the shell.                                                     |
| 85140 |     |                               | Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are                                                |
| 85141 |     |                               | not found.                                                                                                                                          |

85142 **EXAMPLES**

85143 The following command:

85144 `env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile`85145 invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other  
85146 than any variables required by the implementation for conformance. In this case, *PATH* is used  
85147 to locate *mygrep*, which is expected to reside in */mybin*.85148 **RATIONALE**85149 As with all other utilities that invoke other utilities, this volume of POSIX.1-2008 only specifies  
85150 what *env* does with standard input, standard output, standard error, input files, and output files.  
85151 If a utility is executed, it is not constrained by the specification of input and output by *env*.85152 The *-i* option was added to allow the functionality of the removed *-* option in a manner  
85153 compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming  
85154 environment using the *-i* option, as it may remove environment variables required by the  
85155 implementation for conformance. The following will preserve these environment variables as  
85156 well as preserve the *PATH* for conforming utilities:

```

85157 IFS='
85158 '
85159 # The preceding value should be <space><tab><newline>.
85160 # Set IFS to its default value.

85161 set -f
85162 # disable pathname expansion

85163 \unalias -a
85164 # Unset all possible aliases.
85165 # Note that unalias is escaped to prevent an alias
85166 # being used for unalias.
85167 # This step is not strictly necessary, since aliases are not inherited,
85168 # and the ENV environment variable is only used by interactive shells,
85169 # the only way any aliases can exist in a script is if it defines them
85170 # itself.

85171 unset -f env getconf
85172 # Ensure env and getconf are not user functions.

85173 env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command

```

85174 Some have suggested that *env* is redundant since the same effect is achieved by:85175 `name=value ... utility [ argument ... ]`85176 The example is equivalent to *env* when an environment variable is being added to the  
85177 environment of the command, but not when the environment is being set to the given value.  
85178 The *env* utility also writes out the current environment if invoked without arguments. There is  
85179 sufficient functionality beyond what the example provides to justify inclusion of *env*.85180 **FUTURE DIRECTIONS**

85181 None.

85182 **SEE ALSO**85183 [Section 2.14](#) (on page 2334), [Section 2.5](#) (on page 2301)85184 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

85185 **CHANGE HISTORY**

85186 First released in Issue 2.

85187 **Issue 7**85188 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
85189 argument is ' - ' .85190 Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use  
85191 the *env* utility to preserve a conforming environment.

85192 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85193 The EXAMPLES section is revised to change the use of *env* -i.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## 85194 NAME

85195 ex — text editor

## 85196 SYNOPSIS

85197 UP `ex [-rR] [-s|-v] [-c command] [-t tagstring] [-w size] [file...]`

## 85198 DESCRIPTION

85199 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and  
 85200 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**  
 85201 and **visual** commands and in *vi*.

85202 If an operand is *'-'*, the results are unspecified.

85203 This section uses the term *edit buffer* to describe the current working text. No specific  
 85204 implementation is implied by this term. All editing changes are performed on the edit buffer,  
 85205 and no changes to it shall affect any file until an editor command writes the file.

85206 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,  
 85207 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands  
 85208 cannot be supported on such terminals, this condition shall not produce an error message such  
 85209 as “not an editor command” or report a syntax error. The implementation may either accept the  
 85210 commands and produce results on the screen that are the result of an unsuccessful attempt to  
 85211 meet the requirements of this volume of POSIX.1-2008 or report an error describing the terminal-  
 85212 related deficiency.

## 85213 OPTIONS

85214 The *ex* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage  
 85215 of *'-'*, and that *'+'* may be recognized as an option delimiter as well as *'-'*.

85216 The following options shall be supported:

85217 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an  
 85218 existing file (see the EXTENDED DESCRIPTION section). Implementations may  
 85219 support more than a single **-c** option. In such implementations, the specified  
 85220 commands shall be executed in the order specified on the command line.

85221 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery  
 85222 information for a file shall be saved during an editor or system crash (for example,  
 85223 when the editor is terminated by a signal which the editor can catch), or after the  
 85224 use of an *ex* **preserve** command.

85225 A *crash* in this context is an unexpected failure of the system or utility that requires  
 85226 restarting the failed system or utility. A system crash implies that any utilities  
 85227 running at the time also crash. In the case of an editor or system crash, the number  
 85228 of changes to the edit buffer (since the most recent **preserve** command) that will be  
 85229 recovered is unspecified.

85230 If no *file* operands are given and the **-t** option is not specified, all other options, the  
 85231 *EXINIT* variable, and any *.exrc* files shall be ignored; a list of all recoverable files  
 85232 available to the invoking user shall be written, and the editor shall exit normally  
 85233 without further action.

85234 **-R** Set **readonly** edit option.

85235 **-s** Prepare *ex* for batch use by taking the following actions:

- 85236 • Suppress writing prompts and informational (but not diagnostic) messages.
- 85237 • Ignore the value of *TERM* and any implementation default terminal type and
- 85238 assume the terminal is a type incapable of supporting open or visual modes;
- 85239 see the **visual** command and the description of *vi*.
- 85240 • Suppress the use of the *EXINIT* environment variable and the reading of any
- 85241 **.exrc** file; see the EXTENDED DESCRIPTION section.
- 85242 • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 85243 **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The **tags** feature
- 85244 represented by **-t tagstring** and the **tag** command is optional. It shall be provided
- 85245 on any system that also provides a conforming implementation of *ctags*; otherwise,
- 85246 the use of **-t** produces undefined results. On any system, it shall be an error to
- 85247 specify more than a single **-t** option.
- 85248 **-v** Begin in visual mode (see *vi*).
- 85249 **-w size** Set the value of the *window* editor option to *size*.

**OPERANDS**

85250 The following operand shall be supported:

85251 *file* A pathname of a file to be edited.

**STDIN**

85252 The standard input consists of a series of commands and input text, as described in the

85253 EXTENDED DESCRIPTION section. The implementation may limit each line of standard input

85254 to a length of {LINE\_MAX}.

85255 If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.

85256 If a read from the standard input returns an error, or if the editor detects an end-of-file condition

85257 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

**INPUT FILES**

85260 Input files shall be text files or files that would be text files except for an incomplete last line that

85261 is not longer than {LINE\_MAX}-1 bytes in length and contains no NUL characters. By default,

85262 any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other

85263 forms of files may optionally be allowed by *ex* implementations.

85264 The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED

85265 DESCRIPTION section.

85266 By default, the editor shall read lines from the files to be edited without interpreting any of those

85267 lines as any form of editor command.

**ENVIRONMENT VARIABLES**

85270 The following environment variables shall affect the execution of *ex*:

85271 **COLUMNS** Override the system-selected horizontal screen size. See XBD Chapter 8 (on page

85272 173) for valid values and results when it is unset or null.

85273 **EXINIT** Determine a list of *ex* commands that are executed on editor start-up. See the

85274 EXTENDED DESCRIPTION section for more details of the initialization phase.

85275 **HOME** Determine a pathname of a directory that shall be searched for an editor start-up

85276 file named **.exrc**; see the EXTENDED DESCRIPTION section.

|       |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 85277 | <i>LANG</i>                | Provide a default value for the internationalization variables that are unset or null. (See XBD <a href="#">Section 8.2</a> (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                |
| 85278 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85279 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85280 | <i>LC_ALL</i>              | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                          |
| 85281 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85282 | <i>LC_COLLATE</i>          |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85283 |                            | Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.                                                                                                                                                                                                                                                          |
| 85284 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85285 | <i>LC_CTYPE</i>            | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries. |
| 85286 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85287 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85288 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85289 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85290 | <i>LC_MESSAGES</i>         |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85291 |                            | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                      |
| 85292 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85293 | <i>LINES</i>               | Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD <a href="#">Chapter 8</a> (on page 173) for valid values and results when it is unset or null.                                                                                                                                                 |
| 85294 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85295 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85296 | XSI <i>NLSPATH</i>         | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                             |
| 85297 | <i>PATH</i>                | Determine the search path for the shell command specified in the <i>ex</i> editor commands <b>!</b> , <b>shell</b> , <b>read</b> , and <b>write</b> , and the open and visual mode command <b>!</b> ; see the description of command search and execution in <a href="#">Section 2.9.1.1</a> (on page 2317).                                                                                      |
| 85298 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85299 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85300 | <i>SHELL</i>               | Determine the preferred command line interpreter for use as the default value of the <b>shell</b> edit option.                                                                                                                                                                                                                                                                                    |
| 85301 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85302 | <i>TERM</i>                | Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.                                                                                                                                                                                                                                                                   |
| 85303 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85304 | <b>ASYNCHRONOUS EVENTS</b> |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85305 |                            | The following term is used in this and following sections to specify command and asynchronous event actions:                                                                                                                                                                                                                                                                                      |
| 85306 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85307 | <i>complete write</i>      |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85308 |                            | A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the <i>ex</i> <b>preserve</b> command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.                     |
| 85309 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85310 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85311 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85312 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85313 |                            | The following actions shall be taken upon receipt of signals:                                                                                                                                                                                                                                                                                                                                     |
| 85314 | <i>SIGINT</i>              | If the standard input is not a terminal device, <i>ex</i> shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.                                                                                                                                                                                                                           |
| 85315 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85316 |                            | Otherwise, if executing an open or visual text input mode command, <i>ex</i> in receipt of <i>SIGINT</i> shall behave identically to its receipt of the <ESC> character.                                                                                                                                                                                                                          |
| 85317 |                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85318 |                            | Otherwise:                                                                                                                                                                                                                                                                                                                                                                                        |

- 85319 1. If executing an *ex* text input mode command, all input lines that have been  
85320 completely entered shall be resolved into the edit buffer, and any partially  
85321 entered line shall be discarded.
- 85322 2. If there is a currently executing command, it shall be aborted and a message  
85323 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,  
85324 it is unspecified whether any lines modified by the executing command  
85325 appear modified, or as they were before being modified by the executing  
85326 command, in the buffer.
- 85327 If the currently executing command was a motion command, its associated  
85328 command shall be discarded.
- 85329 3. If in open or visual command mode, the terminal shall be alerted.
- 85330 4. The editor shall then return to command mode.
- 85331 **SIGCONT** The screen shall be refreshed if in open or visual mode.
- 85332 **SIGHUP** If the edit buffer has been modified since the last complete write, *ex* shall attempt  
85333 to save the edit buffer so that it can be recovered later using the **-r** option or the *ex*  
85334 **recover** command. The editor shall not write the file or return to command or text  
85335 input mode, and shall terminate with a non-zero exit status.
- 85336 **SIGTERM** Refer to **SIGHUP**.
- 85337 The action taken for all other signals is unspecified.
- 85338 **STDOUT**
- 85339 The standard output shall be used only for writing prompts to the user, for informational  
85340 messages, and for writing lines from the file.
- 85341 **STDERR**
- 85342 The standard error shall be used only for diagnostic messages.
- 85343 **OUTPUT FILES**
- 85344 The output from *ex* shall be text files.
- 85345 **EXTENDED DESCRIPTION**
- 85346 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing  
85347 capabilities available in *ex*.
- 85348 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such  
85349 as inverse video), the message shall be written in standout mode. If the terminal does not  
85350 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the  
85351 error message.
- 85352 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the  
85353 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;  
85354 it can be exited (and command mode re-entered) by typing a <period> (**'.'**) alone at the  
85355 beginning of a line.

85356 **Initialization in ex and vi**

85357 The following symbols are used in this and following sections to specify locations in the edit  
85358 buffer:

85359 *alternate and current pathnames*

85360 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*  
85361 commands that take filenames as arguments shall set them as follows:

- 85362 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**  
85363 command replaces the contents of the edit buffer.
  - 85364 a. If the command replaces the contents of the edit buffer, the *current* pathname  
85365 shall be set to the *file* argument or the file indicated by the tag, and the  
85366 alternate pathname shall be set to the previous value of the *current* pathname.
  - 85367 b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 85368 2. If a *file* argument is specified to the *ex* **next** command:
  - 85369 a. If the command replaces the contents of the edit buffer, the *current* pathname  
85370 shall be set to the first *file* argument, and the alternate pathname shall be set to  
85371 the previous value of the *current* pathname.
- 85372 3. If a *file* argument is specified to the *ex* **file** command, the *current* pathname shall be  
85373 set to the *file* argument, and the alternate pathname shall be set to the previous value  
85374 of the *current* pathname.
- 85375 4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when  
85376 reading or writing a file, and not to the program named by the **shell** edit option), or a  
85377 *file* argument is specified to the *ex* **xit** command:
  - 85378 a. If the *current* pathname has no value, the *current* pathname shall be set to the  
85379 *file* argument.
  - 85380 b. Otherwise, the alternate pathname shall be set to the *file* argument.

85381 If the alternate pathname is set to the previous value of the *current* pathname when the  
85382 *current* pathname had no previous value, then the alternate pathname shall have no value  
85383 as a result.

85384 *current line*

85385 The line of the edit buffer referenced by the cursor. Each command description specifies the  
85386 *current line* after the command has been executed, as the *current line value*. When the edit  
85387 buffer contains no lines, the *current line* shall be zero; see [Addressing in ex](#) (on page 2644).

85388 *current column*

85389 The current display line column occupied by the cursor. (The columns shall be numbered  
85390 beginning at 1.) Each command description specifies the *current column* after the command  
85391 has been executed, as the *current column value*. This column is an *ideal* column that is  
85392 remembered over the lifetime of the editor. The actual display line column upon which the  
85393 cursor rests may be different from the *current column*; see the cursor positioning discussion  
85394 in [Command Descriptions in vi](#) (on page 3310).

85395 *set to non-<blank>*

85396 A description for a *current column* value, meaning that the *current column* shall be set to  
85397 the last display line column on which is displayed any part of the first non-<blank> of the  
85398 line. If the line has no non-<blank> non-<newline> characters, the *current column* shall be  
85399 set to the last display line column on which is displayed any part of the last non-<newline>

- 85400 character in the line. If the line is empty, the current column shall be set to column position  
85401 1.
- 85402 The length of lines in the edit buffer may be limited to {LINE\_MAX} bytes. In open and visual  
85403 mode, the length of lines in the edit buffer may be limited to the number of characters that will  
85404 fit in the display. If either limit is exceeded during editing, an error message shall be written. If  
85405 either limit is exceeded by a line read in from a file, an error message shall be written and the  
85406 edit session may be terminated.
- 85407 If the editor stops running due to any reason other than a user command, and the edit buffer has  
85408 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous  
85409 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.
- 85410 During initialization (before the first file is copied into the edit buffer or any user commands  
85411 from the terminal are processed) the following shall occur:
- 85412 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands  
85413 contained in that variable.
  - 85414 2. If the *EXINIT* variable is not set, and all of the following are true:
    - 85415 a. The *HOME* environment variable is not null and not empty.
    - 85416 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
      - 85417 i. Exists
      - 85418 ii. Is owned by the same user ID as the real user ID of the process or the  
85419 process has appropriate privileges
      - 85420 iii. Is not writable by anyone other than the owner

85421 the editor shall execute the *ex* commands contained in that file.
  - 85422 3. If and only if all of the following are true:
    - 85423 a. The current directory is not referred to by the *HOME* environment variable.
    - 85424 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in  
85425 the directory referred to by the *HOME* environment variable sets the editor option  
85426 **exrc**.
    - 85427 c. The *.exrc* file in the current directory:
      - 85428 i. Exists
      - 85429 ii. Is owned by the same user ID as the real user ID of the process, or by one of  
85430 a set of implementation-defined user IDs
      - 85431 iii. Is not writable by anyone other than the owner

85432 the editor shall attempt to execute the *ex* commands contained in that file.
- 85433 Lines in any *.exrc* file that are blank lines shall be ignored. If any *.exrc* file exists, but is not read  
85434 for ownership or permission reasons, it shall be an error.
- 85435 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user  
85436 shall be edited, as follows:
- 85437 1. If the user specified the *-t* option, the effect shall be as if the *ex tag* command was entered  
85438 with the specified argument, with the exception that if tag processing does not result in a  
85439 file to edit, the effect shall be as described in step 3. below.

- 85440 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if  
85441 the *ex edit* command was entered with the first of those arguments as its *file* argument.
- 85442 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent  
85443 filename as its *file* argument. It is unspecified whether this action shall set the current  
85444 pathname. In an implementation where this action does not set the current pathname, any  
85445 editor command using the current pathname shall fail until an editor command sets the  
85446 current pathname.

85447 If the *-r* option was specified, the first time a file in the initial argument list or a file specified by  
85448 the *-t* option is edited, if recovery information has previously been saved about it, that  
85449 information shall be recovered and the editor shall behave as if the contents of the edit buffer  
85450 have already been modified. If there are multiple instances of the file to be recovered, the one  
85451 most recently saved shall be recovered, and an informational message that there are previous  
85452 versions of the file that can be recovered shall be written. If no recovery information about a file  
85453 is available, an informational message to this effect shall be written, and the edit shall proceed as  
85454 usual.

85455 If the *-c* option was specified, the first time a file that already exists (including a file that might  
85456 not exist but for which recovery information is available, when the *-r* option is specified)  
85457 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of  
85458 the edit buffer, the current column shall be set to non-*<blank>*, and the *ex* commands specified  
85459 with the *-c* option shall be executed. In this case, the current line and current column shall not  
85460 be set as described for the command associated with the replacement or initialization of the edit  
85461 buffer contents. However, if the *-t* option or a *tag* command is associated with this action, the *-c*  
85462 option commands shall be executed and then the movement to the tag shall be performed.

85463 The current argument list shall initially be set to the filenames specified by the user on the  
85464 command line. If no filenames are specified by the user, the current argument list shall be empty.  
85465 If the *-t* option was specified, it is unspecified whether any filename resulting from tag  
85466 processing shall be prepended to the current argument list. In the case where the filename is  
85467 added as a prefix to the current argument list, the current argument list reference shall be set to  
85468 that filename. In the case where the filename is not added as a prefix to the current argument  
85469 list, the current argument list reference shall logically be located before the first of the filenames  
85470 specified on the command line (for example, a subsequent *ex next* command shall edit the first  
85471 filename from the command line). If the *-t* option was not specified, the current argument list  
85472 reference shall be to the first of the filenames on the command line.

### 85473 Addressing in *ex*

85474 Addressing in *ex* relates to the current line and the current column; the address of a line is its  
85475 1-based line number, the address of a column is its 1-based count from the beginning of the line.  
85476 Generally, the current line is the last line affected by a command. The current line number is the  
85477 address of the current line. In each command description, the effect of the command on the  
85478 current line number and the current column is described.

85479 Addresses are constructed as follows:

- 85480 1. The character ' .' (period) shall address the current line.
- 85481 2. The character ' \$' shall address the last line of the edit buffer.
- 85482 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 85483 4. The address " ' x" refers to the line marked with the mark name character ' x', which  
85484 shall be a lowercase letter from the portable character set, the backquote character, or the  
85485 single-quote character. It shall be an error if the line that was marked is not currently

85486 present in the edit buffer or the mark has not been set. Lines can be marked with the *ex*  
85487 **mark** or **k** commands, or the *vi* **m** command.

85488 5. A regular expression enclosed by <slash> characters ('/') shall address the first line  
85489 found by searching forwards from the line following the current line toward the end of  
85490 the edit buffer and stopping at the first line for which the line excluding the terminating  
85491 <newline> matches the regular expression. As stated in [Regular Expressions in ex](#) (on  
85492 page 2675), an address consisting of a null regular expression delimited by <slash>  
85493 characters ("/") shall address the next line for which the line excluding the terminating  
85494 <newline> matches the last regular expression encountered. In addition, the second  
85495 <slash> can be omitted at the end of a command line. If the **wrapsan** edit option is set,  
85496 the search shall wrap around to the beginning of the edit buffer and continue up to and  
85497 including the current line, so that the entire edit buffer is searched. Within the regular  
85498 expression, the sequence "\/" shall represent a literal <slash> instead of the regular  
85499 expression delimiter.

85500 6. A regular expression enclosed in <question-mark> characters ('?') shall address the first  
85501 line found by searching backwards from the line preceding the current line toward the  
85502 beginning of the edit buffer and stopping at the first line for which the line excluding the  
85503 terminating <newline> matches the regular expression. An address consisting of a null  
85504 regular expression delimited by <question-mark> characters ("??") shall address the  
85505 previous line for which the line excluding the terminating <newline> matches the last  
85506 regular expression encountered. In addition, the second <question-mark> can be omitted  
85507 at the end of a command line. If the **wrapsan** edit option is set, the search shall wrap  
85508 around from the beginning of the edit buffer to the end of the edit buffer and continue up  
85509 to and including the current line, so that the entire edit buffer is searched. Within the  
85510 regular expression, the sequence "\?" shall represent a literal <question-mark> instead  
85511 of the RE delimiter.

85512 7. A <plus-sign> ('+') or a minus-sign ('-') followed by a decimal number shall address  
85513 the current line plus or minus the number. A '+' or '-' not followed by a decimal  
85514 number shall address the current line plus or minus 1.

85515 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
85516 Address offsets are constructed as follows:

85517 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the  
85518 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal  
85519 number shall add (subtract) 1 to (from) the address.

85520 2. A decimal number shall add the indicated number of lines to the address.

85521 It shall not be an error for an intermediate address value to be less than zero or greater than the  
85522 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
85523 greater than the last line in the edit buffer.

85524 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in  
85525 [Command Descriptions in ex](#) (on page 2651). If more than the required number of addresses are  
85526 provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than  
85527 the required number of addresses are provided to a command, the addresses specified first shall  
85528 be evaluated and then discarded until the maximum number of valid addresses remain.

85529 Addresses shall be separated from each other by a <comma> (',') or a <semicolon> (;'). If  
85530 no address is specified before or after a <comma> or <semicolon> separator, it shall be as if the  
85531 address of the current line was specified before or after the separator. In the case of a  
85532 <semicolon> separator, the current line ('.') shall be set to the first address, and only then will

85533 the next address be calculated. This feature can be used to determine the starting line for  
85534 forwards and backwards searches (see rules 5. and 6.).

85535 A <percent-sign> ('%') shall be equivalent to entering the two addresses "1, \$".

85536 Any delimiting <blank> characters between addresses, address separators, or address offsets  
85537 shall be discarded.

### 85538 Command Line Parsing in ex

85539 The following symbol is used in this and following sections to describe parsing behavior:

85540 *escape* If a character is referred to as "<backslash>-escaped" or "<control>-V-escaped", it  
85541 shall mean that the character acquired or lost a special meaning by virtue of being  
85542 preceded, respectively, by a <backslash> or <control>-V character. Unless  
85543 otherwise specified, the escaping character shall be discarded at that time and shall  
85544 not be further considered for any purpose.

85545 Command-line parsing shall be done in the following steps. For each step, characters already  
85546 evaluated shall be ignored; that is, the phrase "leading character" refers to the next character  
85547 that has not yet been evaluated.

- 85548 1. Leading <colon> characters shall be skipped.
- 85549 2. Leading <blank> characters shall be skipped.
- 85550 3. If the leading character is a double-quote character, the characters up to and including the  
85551 next non-<backslash>-escaped <newline> shall be discarded, and any subsequent  
85552 characters shall be parsed as a separate command.
- 85553 4. Leading characters that can be interpreted as addresses shall be evaluated; see  
85554 [Addressing in ex](#) (on page 2644).
- 85555 5. Leading <blank> characters shall be skipped.
- 85556 6. If the next character is a <vertical-line> character or a <newline>:  
85557 a. If the next character is a <newline>:  
85558 i. If *ex* is in open or visual mode, the current line shall be set to the last  
85559 address specified, if any.  
85560 ii. Otherwise, if the last command was terminated by a <vertical-line>  
85561 character, no action shall be taken; for example, the command  
85562 "||<newline>" shall execute two implied commands, not three.  
85563 iii. Otherwise, step 6.b. shall apply.
- 85564 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**  
85565 flags specified to any *ex* command shall be remembered and shall apply to this  
85566 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the  
85567 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified  
85568 for that execution of the **number**, **print**, or **list** command.

85569 If *ex* is not currently performing a **global** or **v** command, and no address or count  
85570 is specified, the current line shall be incremented by 1 before the command is  
85571 executed. If incrementing the current line would result in an address past the last  
85572 line in the edit buffer, the command shall fail, and the increment shall not happen.

- 85573 c. The <newline> or <vertical-line> character shall be discarded and any subsequent  
85574 characters shall be parsed as a separate command.
- 85575 7. The command name shall be comprised of the next character (if the character is not  
85576 alphabetic), or the next character and any subsequent alphabetic characters (if the  
85577 character is alphabetic), with the following exceptions:
- 85578 a. Commands that consist of any prefix of the characters in the command name  
85579 **delete**, followed immediately by any of the characters 'l', 'p', '+', '-', or '#'  
85580 shall be interpreted as a **delete** command, followed by a <blank>, followed by the  
85581 characters that were not part of the prefix of the **delete** command. The maximum  
85582 number of characters shall be matched to the command name **delete**, for example,  
85583 "del" shall not be treated as "de" followed by the flag l.
- 85584 b. Commands that consist of the character 'k', followed by a character that can be  
85585 used as the name of a mark, shall be equivalent to the mark command followed by  
85586 a <blank>, followed by the character that followed the 'k'.
- 85587 c. Commands that consist of the character 's', followed by characters that could be  
85588 interpreted as valid options to the s command, shall be the equivalent of the s  
85589 command, without any pattern or replacement values, followed by a <blank>,  
85590 followed by the characters after the 's'.
- 85591 8. The command name shall be matched against the possible command names, and a  
85592 command name that contains a prefix matching the characters specified by the user shall  
85593 be the executed command. In the case of commands where the characters specified by the  
85594 user could be ambiguous, the executed command shall be as follows:

85595  
85596  
85597  
85598  
85599  
85600

|    |        |    |       |    |       |
|----|--------|----|-------|----|-------|
| a  | append | n  | next  | t  | t     |
| c  | change | p  | print | u  | undo  |
| ch | change | pr | print | un | undo  |
| e  | edit   | r  | read  | v  | v     |
| m  | move   | re | read  | w  | write |
| ma | mark   | s  | s     |    |       |

- 85601 Implementation extensions with names causing similar ambiguities shall not be checked  
85602 for a match until all possible matches for commands specified by POSIX.1-2008 have been  
85603 checked.
- 85604 9. If the command is a ! command, or if the command is a **read** command followed by zero  
85605 or more <blank> characters and a !, or if the command is a **write** command followed by  
85606 one or more <blank> characters and a !, the rest of the command shall include all  
85607 characters up to a non-<backslash>-escaped <newline>. The <newline> shall be  
85608 discarded and any subsequent characters shall be parsed as a separate *ex* command.
- 85609 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while  
85610 in open or visual mode, the next part of the command shall be parsed as follows:
- 85611 a. Any '!' character immediately following the command shall be skipped and be  
85612 part of the command.
- 85613 b. Any leading <blank> characters shall be skipped and be part of the command.
- 85614 c. If the next character is a '+', characters up to the first non-<backslash>-escaped  
85615 <newline> or non-<backslash>-escaped <blank> shall be skipped and be part of  
85616 the command.

- 85617 d. The rest of the command shall be determined by the steps specified in paragraph  
85618 12.
- 85619 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the  
85620 command shall be parsed as follows:
- 85621 a. Any leading <blank> characters shall be skipped and be part of the command.
- 85622 b. If the next character is not an alphanumeric, double-quote, <newline>,  
85623 <backslash>, or <vertical-line> character:
- 85624 i. The next character shall be used as a command delimiter.
- 85625 ii. If the command is a **global**, **open**, or **v** command, characters up to the first  
85626 non-<backslash>-escaped <newline>, or first non-<backslash>-escaped  
85627 delimiter character, shall be skipped and be part of the command.
- 85628 iii. If the command is an **s** command, characters up to the first  
85629 non-<backslash>-escaped <newline>, or second non-<backslash>-escaped  
85630 delimiter character, shall be skipped and be part of the command.
- 85631 c. If the command is a **global** or **v** command, characters up to the first  
85632 non-<backslash>-escaped <newline> shall be skipped and be part of the  
85633 command.
- 85634 d. Otherwise, the rest of the command shall be determined by the steps specified in  
85635 paragraph 12.
- 85636 12. Otherwise:
- 85637 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,  
85638 characters up to the first non-<control>-V-escaped <newline>, <vertical-line>, or  
85639 double-quote character shall be skipped and be part of the command.
- 85640 b. Otherwise, characters up to the first non-<backslash>-escaped <newline>,  
85641 <vertical-line>, or double-quote character shall be skipped and be part of the  
85642 command.
- 85643 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.  
85644 ended at a <vertical-line> character, any subsequent characters, up to the next  
85645 non-<backslash>-escaped <newline> shall be used as input text to the command.
- 85646 d. If the command was ended by a double-quote character, all subsequent characters,  
85647 up to the next non-<backslash>-escaped <newline>, shall be discarded.
- 85648 e. The terminating <newline> or <vertical-line> character shall be discarded and any  
85649 subsequent characters shall be parsed as a separate *ex* command.
- 85650 Command arguments shall be parsed as described by the Synopsis and Description of each  
85651 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,  
85652 which must follow the command name without intervening <blank> characters, and where it  
85653 would otherwise be ambiguous. For example, *count* and *flag* arguments need not be  
85654 <blank>-separated because "d22p" is not ambiguous, but *file* arguments to the *ex* **next**  
85655 command must be separated by one or more <blank> characters. Any <blank> in command  
85656 arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be  
85657 <control>-V-escaped, in which case the <blank> shall not be used as an argument delimiter. Any  
85658 <blank> in the command argument for any other command can be <backslash>-escaped, in  
85659 which case that <blank> shall not be used as an argument delimiter.
- 85660 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,

85661 any character can be <control>-V-escaped. All such escaped characters shall be treated literally  
 85662 and shall have no special meaning. Within command arguments for all other *ex* commands that  
 85663 are not regular expressions or replacement strings, any character that would otherwise have a  
 85664 special meaning can be <backslash>-escaped. Escaped characters shall be treated literally,  
 85665 without special meaning as shell expansion characters or `'!'`, `'%'`, and `'#'` expansion  
 85666 characters. See [Regular Expressions in ex](#) (on page 2675) and [Replacement Strings in ex](#) (on page  
 85667 2676) for descriptions of command arguments that are regular expressions or replacement  
 85668 strings.

85669 Non-<backslash>-escaped `'%'` characters appearing in *file* arguments to any *ex* command shall  
 85670 be replaced by the current pathname; unescaped `'#'` characters shall be replaced by the  
 85671 alternate pathname. It shall be an error if `'%'` or `'#'` characters appear unescaped in an  
 85672 argument and their corresponding values are not set.

85673 Non-<backslash>-escaped `'!'` characters in the arguments to either the *ex !* command or the  
 85674 open and visual mode *! command*, or in the arguments to the *ex read* command, where the first  
 85675 non-<blank> after the command name is a `'!'` character, or in the arguments to the *ex write*  
 85676 command where the command name is followed by one or more <blank> characters and the  
 85677 first non-<blank> after the command name is a `'!'` character, shall be replaced with the  
 85678 arguments to the last of those three commands as they appeared after all unescaped `'%'`, `'#'`,  
 85679 and `'!'` characters were replaced. It shall be an error if `'!'` characters appear unescaped in one  
 85680 of these commands and there has been no previous execution of one of these commands.

85681 If an error occurs during the parsing or execution of an *ex* command:

- 85682 • An informational message to this effect shall be written. Execution of the *ex* command shall  
 85683 stop, and the cursor (for example, the current line and column) shall not be further  
 85684 modified.
- 85685 • If the *ex* command resulted from a map expansion, all characters from that map expansion  
 85686 shall be discarded, except as otherwise specified by the **map** command.
- 85687 • Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment  
 85688 variable, a **.exrc** file, a **source** command, a **-c** option, or a **+command** specified to an *ex edit*,  
 85689 **ex next**, or **visual** command, no further commands from the source of the commands shall  
 85690 be executed.
- 85691 • Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v**  
 85692 command, no further commands caused by the execution of the buffer or the **global** or **v**  
 85693 command shall be executed.
- 85694 • Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and  
 85695 including the next non-<backslash>-escaped <newline> shall be discarded.

#### 85696 **Input Editing in ex**

85697 The following symbol is used in this and the following sections to specify command actions:

85698 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and  
 85699 underscores, delimited at both ends by characters other than letters, digits, or  
 85700 underscores, or by the beginning or end of a line or the edit buffer.

85701 When accepting input characters from the user, in either *ex* command mode or *ex* text input  
 85702 mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces  
 85703 volume of POSIX.1-2008.

85704 If in *ex* text input mode:

- 85705 1. If the **number** edit option is set, *ex* shall prompt for input using the line number that  
85706 would be assigned to the line if it is entered, in the format specified for the *ex* **number**  
85707 command.
- 85708 2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters,  
85709 as described by the **autoindent** edit option. **autoindent** characters shall follow the line  
85710 number, if any.

85711 If in *ex* command mode:

- 85712 1. If the **prompt** edit option is set, input shall be prompted for using a single **prompt** character;  
85713 otherwise, there shall be no prompt.

85714 The input characters in the following sections shall have the following effects on the input line.

### 85715 Scroll

85716 *Synopsis:* eof

85717 See the description of the *stty* eof character in *stty*.

85718 If in *ex* command mode:

85719 If the eof character is the first character entered on the line, the line shall be evaluated as if  
85720 it contained two characters: a <control>-D and a <newline>.

85721 Otherwise, the eof character shall have no special meaning.

85722 If in *ex* text input mode:

85723 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be  
85724 modified so that a part of the next text input character will be displayed on the first  
85725 column in the line after the previous **shiftwidth** edit option column boundary, and the  
85726 user shall be prompted again for input for the same line.

85727 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'  
85728 was the previous text input character, the '0' and all **autoindent** characters in the line  
85729 shall be discarded, and the user shall be prompted again for input for the same line.

85730 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'  
85731 was the previous text input character, the '^' and all **autoindent** characters in the line  
85732 shall be discarded, and the user shall be prompted again for input for the same line. In  
85733 addition, the **autoindent** level for the next input line shall be derived from the same line  
85734 from which the **autoindent** level for the current input line was derived.

85735 Otherwise, if there are no **autoindent** or text input characters in the line, the eof character  
85736 shall be discarded.

85737 Otherwise, the eof character shall have no special meaning.

- 85738           **<newline>**
- 85739           *Synopsis:*     <newline>  
85740                            <control>-J
- 85741           If in *ex* command mode:
- 85742                    Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for  
85743                    this purpose.
- 85744           If in *ex* text input mode:
- 85745                    Terminate the current line. If there are no characters other than **autoindent** characters on  
85746                    the line, all characters on the line shall be discarded.
- 85747                    Prompt for text input on a new line after the current line. If the **autoindent** edit option is  
85748                    set, an appropriate number of **autoindent** characters shall be added as a prefix to the line  
85749                    as described by the *ex* **autoindent** edit option.
- 85750           **<backslash>**
- 85751           *Synopsis:*     <backslash>
- 85752            Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any  
85753            special meaning that it may have to the editor during text input mode. The <backslash>  
85754            character shall be retained and evaluated when the command line is parsed, or retained and  
85755            included when the input text becomes part of the edit buffer.
- 85756           **<control>-V**
- 85757           *Synopsis:*     <control>-V
- 85758            Allow the entry of any subsequent character as a literal character, removing any special meaning  
85759            that it may have to the editor during text input mode. The <control>-V character shall be  
85760            discarded before the command line is parsed or the input text becomes part of the edit buffer.
- 85761            If the “literal next” functionality is performed by the underlying system, it is implementation-  
85762            defined whether a character other than <control>-V performs this function.
- 85763           **<control>-W**
- 85764           *Synopsis:*     <control>-W
- 85765            Discard the <control>-W, and the word previous to it in the input line, including any <blank>  
85766            characters following the word and preceding the <control>-W. If the “word erase” functionality  
85767            is performed by the underlying system, it is implementation-defined whether a character other  
85768            than <control>-W performs this function.
- 85769           **Command Descriptions in ex**
- 85770            The following symbols are used in this section to represent command modifiers. Some of these  
85771            modifiers can be omitted, in which case the specified defaults shall be used.
- 85772            *laddr*        A single line address, given in any of the forms described in [Addressing in ex](#) (on  
85773            page 2644); the default shall be the current line (‘.’), unless otherwise specified.
- 85774            If the line address is zero, it shall be an error, unless otherwise specified in the  
85775            following command descriptions.

|       |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 85776 |               | If the edit buffer is empty, and the address is specified with a command other than =, <b>append</b> , <b>insert</b> , <b>open</b> , <b>put</b> , <b>read</b> , or <b>visual</b> , or the address is not zero, it shall be an error.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 85777 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85778 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85779 | <i>2addr</i>  | Two addresses specifying an inclusive range of lines. If no addresses are specified, the default for <i>2addr</i> shall be the current line only (" ., . "), unless otherwise specified in the following command descriptions. If one address is specified, <i>2addr</i> shall specify that line only, unless otherwise specified in the following command descriptions.                                                                                                                                                                                                                                                                                  |
| 85780 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85781 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85782 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85783 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85784 |               | It shall be an error if the first address is greater than the second address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 85785 |               | If the edit buffer is empty, and the two addresses are specified with a command other than the <b>!</b> , <b>write</b> , <b>wq</b> , or <b>xit</b> commands, or either address is not zero, it shall be an error.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 85786 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85787 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85788 | <i>count</i>  | A positive decimal number. If <i>count</i> is specified, it shall be equivalent to specifying an additional address to the command, unless otherwise specified by the following command descriptions. The additional address shall be equal to the last address specified to the command (either explicitly or by default) plus <i>count</i> −1.                                                                                                                                                                                                                                                                                                          |
| 85789 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85790 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85791 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85792 |               | If this would result in an address greater than the last line of the edit buffer, it shall be corrected to equal the last line of the edit buffer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 85793 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85794 | <i>flags</i>  | One or more of the characters '+', '-', '#', 'p', or 'l' (ell). The flag characters can be <blank>-separated, and in any order or combination. The characters '#', 'p', and 'l' shall cause lines to be written in the format specified by the <b>print</b> command with the specified <i>flags</i> .                                                                                                                                                                                                                                                                                                                                                     |
| 85795 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85796 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85797 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85798 |               | The lines to be written are as follows:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 85799 |               | 1. All edit buffer lines written during the execution of the <i>ex</i> <b>&amp;</b> , <b>~</b> , <b>list</b> , <b>number</b> , <b>open</b> , <b>print</b> , <b>s</b> , <b>visual</b> , and <b>z</b> commands shall be written as specified by <i>flags</i> .                                                                                                                                                                                                                                                                                                                                                                                              |
| 85800 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85801 |               | 2. After the completion of an <i>ex</i> command with a flag as an argument, the current line shall be written as specified by <i>flags</i> , unless the current line was the last line written by the command.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 85802 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85803 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85804 |               | The characters '+' and '-' cause the value of the current line after the execution of the <i>ex</i> command to be adjusted by the offset address as described in <a href="#">Addressing in ex</a> (on page 2644). This adjustment shall occur before the current line is written as described in 2. above.                                                                                                                                                                                                                                                                                                                                                |
| 85805 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85806 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85807 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85808 |               | The default for <i>flags</i> shall be none.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 85809 | <i>buffer</i> | One of a number of named areas for holding text. The named buffers are specified by the alphanumeric characters of the POSIX locale. There shall also be one "unnamed" buffer. When no buffer is specified for editor commands that use a buffer, the unnamed buffer shall be used. Commands that store text into buffers shall store the text as it was before the command took effect, and shall store text occurring earlier in the file before text occurring later in the file, regardless of how the text region was specified. Commands that store text into buffers shall store the text into the unnamed buffer as well as any specified buffer. |
| 85810 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85811 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85812 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85813 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85814 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85815 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85816 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85817 |               | In <i>ex</i> commands, buffer names are specified as the name by itself. In open or visual mode commands the name is preceded by a double-quote (' "') character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 85818 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 85819 |               | If the specified buffer name is an uppercase character, and the buffer contents are to be modified, the buffer shall be appended to rather than being overwritten. If                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 85820 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

- 85821 the buffer is not being modified, specifying the buffer name in lowercase and  
85822 uppercase shall have identical results.
- 85823 There shall also be buffers named by the numbers 1 through 9. In open and visual  
85824 mode, if a region of text including characters from more than a single line is being  
85825 modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or  
85826 **d** commands specifies that the buffer text shall be in line mode, or the commands  
85827 **%**, **'**, **/**, **?**, **(**, **)**, **N**, **n**, **{**, or **}** are used to define a region of text for the **c** or **d** commands,  
85828 the contents of buffers 1 through 8 shall be moved into the buffer named by the  
85829 next numerically greater value, the contents of buffer 9 shall be discarded, and the  
85830 region of text shall be copied into buffer 1. This shall be in addition to copying the  
85831 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can  
85832 be specified as a source buffer for open and visual mode commands; however,  
85833 specifying a numeric buffer as the write target of an open or visual mode  
85834 command shall have unspecified results.
- 85835 The text of each buffer shall have the characteristic of being in either line or  
85836 character mode. Appending text to a non-empty buffer shall set the mode to match  
85837 the characteristic of the text being appended. Appending text to a buffer shall  
85838 cause the creation of at least one additional line in the buffer. All text stored into  
85839 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers  
85840 as the source of text specify individually how buffers of different modes are  
85841 handled. Each open or visual mode command that uses buffers for any purpose  
85842 specifies individually the mode of the text stored into the buffer and how buffers  
85843 of different modes are handled.
- 85844 *file* Command text used to derive a pathname. The default shall be the current  
85845 pathname, as defined previously, in which case, if no current pathname has yet  
85846 been established it shall be an error, except where specifically noted in the  
85847 individual command descriptions that follow. If the command text contains any of  
85848 the characters **'** **~** **'** **{** **'** **[** **'** **\*** **'** **?** **'** **\$** **'** **"** **'**, backquote, single-quote, and  
85849 **<backslash>**, it shall be subjected to the process of "shell expansions", as described  
85850 below; if more than a single pathname results and the command expects only one,  
85851 it shall be an error.
- 85852 The process of shell expansions in the editor shall be done as follows. The *ex* utility  
85853 shall pass two arguments to the program named by the shell edit option; the first  
85854 shall be **-c**, and the second shall be the string "echo" and the command text as a  
85855 single argument. The standard output and standard error of that command shall  
85856 replace the command text.
- 85857 **!** A character that can be appended to the command name to modify its operation,  
85858 as detailed in the individual command descriptions. With the exception of the *ex*  
85859 **read**, **write**, and **!** commands, the **' !'** character shall only act as a modifier if there  
85860 are no **<blank>** characters between it and the command name.
- 85861 *remembered search direction*  
85862 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in  
85863 the edit buffer based on a remembered search direction, which is initially unset,  
85864 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

85865 **Abbreviate**85866 *Synopsis:* `ab[breviate] [lhs rhs]`85867 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.85868 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
85869 characters and <blank> characters shall not be restricted. Additional restrictions shall be  
85870 implementation-defined.85871 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character  
85872 shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.85873 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a  
85874 <control>-V character is entered after a word character, a check shall be made for a set of  
85875 characters matching *lhs*, in the text input entered during this command. If it is found, the effect  
85876 shall be as if *rhs* was entered instead of *lhs*.

85877 The set of characters that are checked is defined as follows:

- 85878 1. If there are no characters inserted before the word and non-word or <ESC> characters  
85879 that triggered the check, the set of characters shall consist of the word character.
- 85880 2. If the character inserted before the word and non-word or <ESC> characters that  
85881 triggered the check is a word character, the set of characters shall consist of the characters  
85882 inserted immediately before the triggering characters that are word characters, plus the  
85883 triggering word character.
- 85884 3. If the character inserted before the word and non-word or <ESC> characters that  
85885 triggered the check is not a word character, the set of characters shall consist of the  
85886 characters that were inserted before the triggering characters that are neither <blank>  
85887 characters nor word characters, plus the triggering word character.

85888 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**  
85889 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the  
85890 effect of the command shall be as if the replacement had not occurred.85891 *Current line:* Unchanged.85892 *Current column:* Unchanged.85893 **Append**85894 *Synopsis:* `[laddr] a[ppend] [!]`85895 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is  
85896 specified, the text shall be placed at the beginning of the edit buffer.85897 This command shall be affected by the **number** and **autoindent** edit options; following the  
85898 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
85899 duration of this command only.85900 *Current line:* Set to the last input line; if no lines were input, set to the specified line, or to the first  
85901 line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.85902 *Current column:* Set to non-<blank>.

85903 **Arguments**85904 *Synopsis:* ar[gs]85905 Write the current argument list, with the current argument-list entry, if any, between '[' and  
85906 ']' characters.85907 *Current line:* Unchanged.85908 *Current column:* Unchanged.85909 **Change**85910 *Synopsis:* [2addr] c[hange][!][count]85911 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall  
85912 be copied into the unnamed buffer, which shall become a line mode buffer.85913 This command shall be affected by the **number** and **autoindent** edit options; following the  
85914 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
85915 duration of this command only.85916 *Current line:* Set to the last input line; if no lines were input, set to the line before the first  
85917 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to  
85918 zero if the edit buffer is empty.85919 *Current column:* Set to non-<blank>.85920 **Change Directory**85921 *Synopsis:* chd[ir][!][directory]

85922 cd[!][directory]

85923 Change the current working directory to *directory*.85924 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null  
85925 and non-empty value, *directory* shall default to the value named in the *HOME* environment  
85926 variable. If the *HOME* environment variable is empty or is undefined, the default value of  
85927 *directory* is implementation-defined.85928 If no '!' is appended to the command name, and the edit buffer has been modified since the  
85929 last complete write, and the current pathname does not begin with a '/', it shall be an error.85930 *Current line:* Unchanged.85931 *Current column:* Unchanged.85932 **Copy**85933 *Synopsis:* [2addr] co[py] laddr [flags]

85934 [2addr] t laddr [flags]

85935 Copy the specified lines after the specified destination line; line zero specifies that the lines shall  
85936 be placed at the beginning of the edit buffer.85937 *Current line:* Set to the last line copied.85938 *Current column:* Set to non-<blank>.

85939 **Delete**85940 *Synopsis:* `[2addr] d[etele][buffer][count][flags]`85941 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a  
85942 line-mode buffer.85943 Flags can immediately follow the command name; see [Command Line Parsing in ex](#) (on page  
85944 2646).85945 *Current line:* Set to the line following the deleted lines, or to the last line in the edit buffer if that  
85946 line is past the end of the edit buffer, or to zero if the edit buffer is empty.85947 *Current column:* Set to non-<blank>.85948 **Edit**85949 *Synopsis:* `e[dit][!][+command][file]`85950 `ex[!][+command][file]`85951 If no '!' is appended to the command name, and the edit buffer has been modified since the  
85952 last complete write, it shall be an error.85953 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,  
85954 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the  
85955 edit buffer with the current contents of the file named by the current pathname. If for any reason  
85956 the current contents of the file cannot be accessed, the edit buffer shall be empty.85957 The *+command* option shall be <blank>-delimited; <blank> characters within the *+command* can  
85958 be escaped by preceding them with a <backslash> character. The *+command* shall be interpreted  
85959 as an *ex* command immediately after the contents of the edit buffer have been replaced and the  
85960 current line and column have been set.

85961 If the edit buffer is empty:

85962 *Current line:* Set to 0.85963 *Current column:* Set to 1.85964 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:85965 *Current line:* Set to the last line of the edit buffer.85966 *Current column:* Set to non-<blank>.85967 Otherwise, if *file* is omitted or results in the current pathname:85968 *Current line:* Set to the first line of the edit buffer.85969 *Current column:* Set to non-<blank>.85970 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if  
85971 the file was previously edited, the line and column may be set as follows:85972 *Current line:* Set to the last value held when that file was last edited. If this value is not a valid  
85973 line in the new edit buffer, set to the first line of the edit buffer.85974 *Current column:* If the current line was set to the last value held when the file was last edited, set  
85975 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid  
85976 column in the new edit buffer, set to non-<blank>.

85977 Otherwise:

85978 *Current line*: Set to the first line of the edit buffer.

85979 *Current column*: Set to non-<blank>.

## 85980 File

85981 *Synopsis*: `f[file] [file]`

85982 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and  
85983 the current pathname shall be set to *file*.

85984 Write an informational message. If the file has a current pathname, it shall be included in this  
85985 message; otherwise, the message shall indicate that there is no current pathname. If the edit  
85986 buffer contains lines, the current line number and the number of lines in the edit buffer shall be  
85987 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If  
85988 the edit buffer has been modified since the last complete write, this fact shall be included in this  
85989 message. If the **readonly** edit option is set, this fact shall be included in this message. The  
85990 message may contain other unspecified information.

85991 *Current line*: Unchanged.

85992 *Current column*: Unchanged.

## 85993 Global

85994 *Synopsis*: `[2addr] g[lobal] /pattern/ [commands]`  
85995 `[2addr] v /pattern/ [commands]`

85996 The optional **'!** character after the **global** command shall be the same as executing the **v**  
85997 command.

85998 If *pattern* is empty (for example, `"/*"`) or not specified, the last regular expression used in the  
85999 editor command shall be used as the *pattern*. The *pattern* can be delimited by <slash> characters  
86000 (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than  
86001 <backslash>, <vertical-line>, <newline>, or double-quote.

86002 If no lines are specified, the lines shall default to the entire file.

86003 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the  
86004 specified lines for which the line excluding the terminating <newline> matches (**global**) or does  
86005 not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by  
86006 *commands*, with the current line (`'.'`) set to each marked line. If an error occurs during this  
86007 process or the contents of the edit buffer are replaced (for example, by the *ex* **edit** command) an  
86008 error message shall be written and no more commands resulting from the execution of this  
86009 command shall be processed.

86010 Multiple *ex* commands can be specified by entering multiple commands on a single line using a  
86011 <vertical-line> to delimit them, or one per line, by escaping each <newline> with a <backslash>.

86012 If no commands are specified:

- 86013 1. If in *ex* command mode, it shall be as if the **print** command were specified.
- 86014 2. Otherwise, no command shall be executed.

86015 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the  
86016 command, and the terminating <period> can be omitted if the command ends the list of  
86017 commands. The **open** and **visual** commands can be specified as one of the commands, in which

86018 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode  
86019 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open  
86020 or visual mode reentered, until the list of marked lines is exhausted.

86021 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted  
86022 by commands executed for lines occurring earlier in the file than the marked lines. In this case,  
86023 no commands shall be executed for the deleted lines.

86024 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

86025 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**  
86026 command.

86027 *Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for the  
86028 executed *ex* commands.

86029 *Current column*: If no commands are executed, set to non-<blank>; otherwise, as specified for the  
86030 individual *ex* commands.

### 86031 **Insert**

86032 *Synopsis*: `[1addr] i[nsert][!]`

86033 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero  
86034 or 1, the text shall be placed at the beginning of the edit buffer.

86035 This command shall be affected by the **number** and **autoindent** edit options; following the  
86036 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
86037 duration of this command only.

86038 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified  
86039 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero  
86040 if the edit buffer is empty.

86041 *Current column*: Set to non-<blank>.

### 86042 **Join**

86043 *Synopsis*: `[2addr] j[oin][!][count][flags]`

86044 If *count* is specified:

86045 If no address was specified, the **join** command shall behave as if *2addr* were the current  
86046 line and the current line plus *count* (*.,. + count*).

86047 If one address was specified, the **join** command shall behave as if *2addr* were the specified  
86048 address and the specified address plus *count* (*addr,addr + count*).

86049 If two addresses were specified, the **join** command shall behave as if an additional  
86050 address, equal to the last address plus *count* -1 (*addr1,addr2,addr2 + count -1*), was  
86051 specified.

86052 If this would result in a second address greater than the last line of the edit buffer, it shall  
86053 be corrected to be equal to the last line of the edit buffer.

86054 If no *count* is specified:

86055 If no address was specified, the **join** command shall behave as if *2addr* were the current  
86056 line and the next line (*.,. +1*).

86057 If one address was specified, the **join** command shall behave as if *2addr* were the specified  
86058 address and the next line (*addr,addr +1*).

86059 Join the text from the specified lines together into a single line, which shall replace the specified  
86060 lines.

86061 If a **' !'** character is appended to the command name, the **join** shall be without modification of  
86062 any line, independent of the current locale.

86063 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for  
86064 each subsequent line, proceed as follows:

- 86065 1. Discard leading <space> characters from the line to be joined.
- 86066 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 86067 3. If the current line ends in a <blank>, or the first character of the line to be joined is a **' )'**  
86068 character, join the lines without further modification.
- 86069 4. If the last character of the current line is a **' . '**, join the lines with two <space> characters  
86070 between them.
- 86071 5. Otherwise, join the lines with a single <space> between them.

86072 *Current line*: Set to the first line specified.

86073 *Current column*: Set to non-<blank>.

#### 86074 **List**

86075 *Synopsis*: `[2addr] l[ist][count][flags]`

86076 This command shall be equivalent to the *ex* command:

86077 `[2addr] p[rint][count] l[flags]`

86078 See [Print](#) (on page 2663).

#### 86079 **Map**

86080 *Synopsis*: `map[!][lhs rhs]`

86081 If *lhs* and *rhs* are not specified:

- 86082 1. If **' !'** is specified, write the current list of text input mode maps.
- 86083 2. Otherwise, write the current list of command mode maps.
- 86084 3. Do nothing more.

86085 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
86086 characters and <blank> characters shall not be restricted. Additional restrictions shall be  
86087 implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in  
86088 which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V  
86089 shall be discarded.

86090 If the character **' !'** is appended to the **map** command name, the mapping shall be effective  
86091 during open or visual text input mode rather than **open** or **visual** command mode. This allows  
86092 *lhs* to have two different **map** definitions at the same time: one for command mode and one for

- 86093 text input mode.
- 86094 For command mode mappings:
- 86095 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as  
86096 part of the arguments to the command), the action shall be as if the corresponding *rhs* had  
86097 been entered.
- 86098 If any character in the command, other than the first, is escaped using a <control>-V  
86099 character, that character shall not be part of a match to an *lhs*.
- 86100 It is unspecified whether implementations shall support **map** commands where the *lhs* is  
86101 more than a single character in length, where the first character of the *lhs* is printable.
- 86102 If *lhs* contains more than one character and the first character is '#', followed by a  
86103 sequence of digits corresponding to a numbered function key, then when this function key  
86104 is typed it shall be mapped to *rhs*. Characters other than digits following a '#' character  
86105 also represent the function key named by the characters in the *lhs* following the '#' and  
86106 may be mapped to *rhs*. It is unspecified how function keys are named or what function  
86107 keys are supported.
- 86108 For text input mode mappings:
- 86109 When the *lhs* is entered as any part of text entered in open or visual text input modes, the  
86110 action shall be as if the corresponding *rhs* had been entered.
- 86111 If any character in the input text is escaped using a <control>-V character, that character  
86112 shall not be part of a match to an *lhs*.
- 86113 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is  
86114 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or  
86115 not the display appears as if the corresponding *rhs* text was entered, the effect of the  
86116 command shall be as if the *lhs* text was entered.
- 86117 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,  
86118 possibly matching characters before treating the already entered characters as not matching the  
86119 *lhs*.
- 86120 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the  
86121 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those  
86122 characters shall not be remapped.
- 86123 On block-mode terminals, the mapping need not occur immediately (for example, it may occur  
86124 after the terminal transmits a group of characters to the system), but it shall achieve the same  
86125 results as if it occurred immediately.
- 86126 *Current line*: Unchanged.
- 86127 *Current column*: Unchanged.

86128 **Mark**86129 *Synopsis:* `[laddr] ma[rk] character`86130 `[laddr] k character`

86131 Implementations shall support *character* values of a single lowercase letter of the POSIX locale  
 86132 and the backquote and single-quote characters; support of other characters is implementation-  
 86133 defined.

86134 If executing the *vi m* command, set the specified mark to the current line and 1-based numbered  
 86135 character referenced by the current column, if any; otherwise, column position 1.

86136 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>  
 86137 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;  
 86138 otherwise, column position 1.

86139 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a  
 86140 deleted line is restored by a subsequent **undo** command, any marks previously associated with  
 86141 the line, which have not been reset, shall be restored as well. Any use of a mark not associated  
 86142 with a current line in the edit buffer shall be an error.

86143 The marks ' and ' shall be set as described previously, immediately before the following events  
 86144 occur in the editor:

- 86145 1. The use of '\$' as an *ex* address
- 86146 2. The use of a positive decimal number as an *ex* address
- 86147 3. The use of a search command as an *ex* address
- 86148 4. The use of a mark reference as an *ex* address
- 86149 5. The use of the following open and visual mode commands: <control>-], %, (, ), [, ], {, }
- 86150 6. The use of the following open and visual mode commands: ', G, H, L, M, z if the current  
 86151 line will change as a result of the command
- 86152 7. The use of the open and visual mode commands: /, ?, N, ', n if the current line or column  
 86153 will change as a result of the command
- 86154 8. The use of the *ex* mode commands: z, **undo**, **global**, v

86155 For rules 1., 2., 3., and 4., the ' and ' marks shall not be set if the *ex* command is parsed as  
 86156 specified by rule 6.a. in [Command Line Parsing in ex](#) (on page 2646).

86157 For rules 5., 6., and 7., the ' and ' marks shall not be set if the commands are used as motion  
 86158 commands in open and visual mode.

86159 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ' and ' marks shall not be set if the command fails.

86160 The ' and ' marks shall be set as described previously, each time the contents of the edit buffer  
 86161 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex**  
 86162 mode and the edit buffer is not empty, before any commands or movements (including  
 86163 commands or movements specified by the **-c** or **-t** options or the **+command** argument) are  
 86164 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi*  
 86165 **m** command; otherwise, as if executing the *ex mark* command.

86166 When changing from **ex** mode to open or visual mode, if the ' and ' marks are not already set,  
 86167 the ' and ' marks shall be set as described previously.

86168 *Current line:* Unchanged.

86169 *Current column*: Unchanged.

### 86170 **Move**

86171 *Synopsis*: `[2addr] m[ove] laddr [flags]`

86172 Move the specified lines after the specified destination line. A destination of line zero specifies  
86173 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the  
86174 destination line is within the range of lines to be moved.

86175 *Current line*: Set to the last of the moved lines.

86176 *Current column*: Set to non-<blank>.

### 86177 **Next**

86178 *Synopsis*: `n[ext] [!] [+command] [file ...]`

86179 If no '!' is appended to the command name, and the edit buffer has been modified since the  
86180 last complete write, it shall be an error, unless the file is successfully written as specified by the  
86181 **autowrite** option.

86182 If one or more files is specified:

- 86183 1. Set the argument list to the specified filenames.
- 86184 2. Set the current argument list reference to be the first entry in the argument list.
- 86185 3. Set the current pathname to the first filename specified.

86186 Otherwise:

- 86187 1. It shall be an error if there are no more filenames in the argument list after the filename  
86188 currently referenced.
- 86189 2. Set the current pathname and the current argument list reference to the filename after the  
86190 filename currently referenced in the argument list.

86191 Replace the contents of the edit buffer with the contents of the file named by the current  
86192 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
86193 empty.

86194 This command shall be affected by the **autowrite** and **writeany** edit options.

86195 The *+command* option shall be <blank>-delimited; <blank> characters can be escaped by  
86196 preceding them with a <backslash> character. The *+command* shall be interpreted as an *ex*  
86197 command immediately after the contents of the edit buffer have been replaced and the current  
86198 line and column have been set.

86199 *Current line*: Set as described for the **edit** command.

86200 *Current column*: Set as described for the **edit** command.

86201 **Number**86202 *Synopsis:* [2addr] nu[mber] [count] [flags]

86203 [2addr] #[count] [flags]

86204 These commands shall be equivalent to the *ex* command:

86205 [2addr] p[rint] [count] #[flags]

86206 See [Print](#).86207 **Open**86208 *Synopsis:* [1addr] o[pen] /pattern/ [flags]

86209 This command need not be supported on block-mode terminals or terminals with insufficient  
 86210 capabilities. If standard input, standard output, or standard error are not terminal devices, the  
 86211 results are unspecified.

86212 Enter open mode.

86213 The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is  
 86214 empty (for example, "//") or not specified, the last regular expression used in the editor shall  
 86215 be used as the pattern. The pattern can be delimited by <slash> characters (shown in the  
 86216 Synopsis), as well as any alphanumeric, or non-<blank> other than <backslash>, <vertical-line>,  
 86217 <newline>, or double-quote.

86218 *Current line:* Set to the specified line.86219 *Current column:* Set to non-<blank>.86220 **Preserve**86221 *Synopsis:* pre[serve]

86222 Save the edit buffer in a form that can later be recovered by using the `-r` option or by using the  
 86223 *ex* **recover** command. After the file has been preserved, a mail message shall be sent to the user.  
 86224 This message shall be readable by invoking the *mailx* utility. The message shall contain the name  
 86225 of the file, the time of preservation, and an *ex* command that could be used to recover the file.  
 86226 Additional information may be included in the mail message.

86227 *Current line:* Unchanged.86228 *Current column:* Unchanged.86229 **Print**86230 *Synopsis:* [2addr] p[rint] [count] [flags]

86231 Write the addressed lines. The behavior is unspecified if the number of columns on the display is  
 86232 less than the number of columns required to write any single character in the lines being written.

86233 Non-printable characters, except for the <tab>, shall be written as implementation-defined  
 86234 multi-character sequences.

86235 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line  
 86236 number in the following format:

86237 "%6dΔΔ", &lt;line number&gt;

86238 If the **l** flag is specified or the **list** edit option is set:

- 86239 1. The characters listed in XBD Table 5-1 (on page 121) shall be written as the corresponding  
86240 escape sequence.
- 86241 2. Non-printable characters not in XBD Table 5-1 (on page 121) shall be written as one three-  
86242 digit octal number (with a preceding <backslash>) for each byte in the character (most  
86243 significant byte first).
- 86244 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line  
86245 shall be written with a preceding <backslash>.

86246 Long lines shall be folded; the length at which folding occurs is unspecified, but should be  
86247 appropriate for the output terminal, considering the number of columns of the terminal.

86248 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified  
86249 whether a multi-column character at the folding position is separated; it shall not be discarded.

86250 *Current line*: Set to the last written line.

86251 *Current column*: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

### 86252 Put

86253 *Synopsis*: `[laddr] pu[t][buffer]`

86254 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line  
86255 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a  
86256 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

86257 *Current line*: Set to the last line entered into the edit buffer.

86258 *Current column*: Set to non-<blank>.

### 86259 Quit

86260 *Synopsis*: `q[uit][!]`

86261 If no '!' is appended to the command name:

- 86262 1. If the edit buffer has been modified since the last complete write, it shall be an error.
- 86263 2. If there are filenames in the argument list after the filename currently referenced, and the  
86264 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit**, on page 3344) command, it shall be  
86265 an error.

86266 Otherwise, terminate the editing session.

### 86267 Read

86268 *Synopsis*: `[laddr] r[ead][!][file]`

86269 If '!' is not the first non-<blank> to follow the command name, a copy of the specified file shall  
86270 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be  
86271 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If  
86272 no *file* is named, the current pathname shall be the default. If there is no current pathname, then  
86273 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be  
86274 an error. Specifying a *file* that is not of type regular shall have unspecified results.

86275 Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and  
86276 '!' characters expanded as described in **Command Line Parsing in ex** (on page 2646).

86277 The *ex* utility shall then pass two arguments to the program named by the shell edit option; the

86278 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a  
 86279 single argument. The standard input of the program shall be set to the standard input of the *ex*  
 86280 program when it was invoked. The standard error and standard output of the program shall be  
 86281 appended into the edit buffer after the specified line.

86282 Each line in the copied file or program output (as delimited by `<newline>` characters or the end  
 86283 of the file or output if it is not immediately preceded by a `<newline>`), shall be a separate line in  
 86284 the edit buffer. Any occurrences of `<carriage-return>` and `<newline>` pairs in the output shall be  
 86285 treated as single `<newline>` characters.

86286 The special meaning of the `' !'` following the **read** command can be overridden by escaping it  
 86287 with a `<backslash>` character.

86288 *Current line*: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual  
 86289 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into  
 86290 the edit buffer.

86291 *Current column*: Set to non-`<blank>`.

## 86292 **Recover**

86293 *Synopsis*: `rec[over] [!] file`

86294 If no `' !'` is appended to the command name, and the edit buffer has been modified since the  
 86295 last complete write, it shall be an error.

86296 If no *file* operand is specified, then the current pathname shall be used. If there is no current  
 86297 pathname or *file* operand, it shall be an error.

86298 If no recovery information has previously been saved about *file*, the **recover** command shall  
 86299 behave identically to the **edit** command, and an informational message to this effect shall be  
 86300 written.

86301 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer  
 86302 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the  
 86303 one most recently saved shall be recovered, and an informational message that there are  
 86304 previous versions of the file that can be recovered shall be written. The editor shall behave as if  
 86305 the contents of the edit buffer have already been modified.

86306 *Current file*: Set as described for the **edit** command.

86307 *Current column*: Set as described for the **edit** command.

## 86308 **Rewind**

86309 *Synopsis*: `rew[ind] [!]`

86310 If no `' !'` is appended to the command name, and the edit buffer has been modified since the  
 86311 last complete write, it shall be an error, unless the file is successfully written as specified by the  
 86312 **autowrite** option.

86313 If the argument list is empty, it shall be an error.

86314 The current argument list reference and the current pathname shall be set to the first filename in  
 86315 the argument list.

86316 Replace the contents of the edit buffer with the contents of the file named by the current  
 86317 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
 86318 empty.

86319 This command shall be affected by the **autowrite** and **writeany** edit options.

86320 *Current line*: Set as described for the **edit** command.

86321 *Current column*: Set as described for the **edit** command.

### 86322 **Set**

86323 *Synopsis*: `se[t][option]=[value] ...][nooption ...][option? ...][all]`

86324 When no arguments are specified, write the value of the **term** edit option and those options  
86325 whose values have been changed from the default settings; when the argument *all* is specified,  
86326 write all of the option values.

86327 Giving an option name followed by the character '?' shall cause the current value of that  
86328 option to be written. The '?' can be separated from the option name by zero or more <blank>  
86329 characters. The '?' shall be necessary only for Boolean valued options. Boolean options can be  
86330 given values by the form **set option** to turn them on or **set nooption** to turn them off; string and  
86331 numeric options can be assigned by the form **set option=value**. Any <blank> characters in strings  
86332 can be included as is by preceding each <blank> with an escaping <backslash>. More than one  
86333 option can be set or listed by a single set command by specifying multiple arguments, each  
86334 separated from the next by one or more <blank> characters.

86335 See [Edit Options in ex](#) (on page 2676) for details about specific options.

86336 *Current line*: Unchanged.

86337 *Current column*: Unchanged.

### 86338 **Shell**

86339 *Synopsis*: `sh[ell]`

86340 Invoke the program named in the **shell** edit option with the single argument **-i** (interactive  
86341 mode). Editing shall be resumed when the program exits.

86342 *Current line*: Unchanged.

86343 *Current column*: Unchanged.

### 86344 **Source**

86345 *Synopsis*: `so[urce] file`

86346 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

86347 *Current line*: As specified for the individual *ex* commands.

86348 *Current column*: As specified for the individual *ex* commands.

### 86349 **Substitute**

86350 *Synopsis*: `[2addr] s[substitute] [/pattern/repl/ [options] [count] [flags]]`  
86351 `[2addr] & [options] [count] [flags]`  
86352 `[2addr] ~ [options] [count] [flags]`

86353 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See  
86354 [Regular Expressions in ex](#) (on page 2675) and [Replacement Strings in ex](#) (on page 2676).) Any  
86355 non-alphabetic, non-<blank> delimiter other than <backslash>, '|', <newline>, or double-  
86356 quote can be used instead of '/'. <backslash> characters can be used to escape delimiters,  
86357 <backslash> characters, and other special characters.

- 86358 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If  
 86359 both *pattern* and *repl* are not specified or are empty (for example, "//"), the last **s** command  
 86360 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in  
 86361 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be  
 86362 replaced by nothing. If the entire replacement pattern is '%', the last replacement pattern to an  
 86363 **s** command shall be used.
- 86364 Entering a <carriage-return> in *repl* (which requires an escaping <backslash> in *ex* mode and an  
 86365 escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in  
 86366 the edit buffer. The <carriage-return> shall be discarded.
- 86367 If *options* includes the letter 'g' (**global**), all non-overlapping instances of the pattern in the line  
 86368 shall be replaced.
- 86369 If *options* includes the letter 'c' (**confirm**), then before each substitution the line shall be written;  
 86370 the written line shall reflect all previous substitutions. On the following line, <space> characters  
 86371 shall be written beneath the characters from the line that are before the *pattern* to be replaced,  
 86372 and '^' characters written beneath the characters included in the *pattern* to be replaced. The *ex*  
 86373 utility shall then wait for a response from the user. An affirmative response shall cause the  
 86374 substitution to be done, while any other input shall not make the substitution. An affirmative  
 86375 response shall consist of a line with the affirmative response (as defined by the current locale) at  
 86376 the beginning of the line. This line shall be subject to editing in the same way as the *ex* command  
 86377 line.
- 86378 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the  
 86379 user shall be preserved in the edit buffer after the interrupt.
- 86380 If the remembered search direction is not set, the **s** command shall set it to forward.
- 86381 In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&**  
 86382 command were replaced by:
- 86383 *s/pattern/repl/*
- 86384 where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.
- 86385 In the third Synopsis, the **~** command shall repeat the previous substitution, as if the '**~**' were  
 86386 replaced by:
- 86387 *s/pattern/repl/*
- 86388 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from  
 86389 the previous substitution (including **&** and **~**) command.
- 86390 These commands shall be affected by the *LC\_MESSAGES* environment variable.
- 86391 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no substitution  
 86392 occurred.
- 86393 *Current column*: Set to non-<blank>.

86394 **Suspend**

86395 *Synopsis:*    su[spend][!]  
86396            st[op][!]

86397 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the  
86398 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell  
86399 (see the description of *set -m*).

86400 These commands shall be affected by the **autowrite** and **writeany** edit options.

86401 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

86402 **Tag**

86403 *Synopsis:*    ta[g][!] *tagstring*

86404 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see  
86405 *ctags*) description.

86406 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in  
86407 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from  
86408 beginning to end. If no reference is found, it shall be an error and an error message to this effect  
86409 shall be written. If the reference is not found, or if an error occurs while processing a file referred  
86410 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first  
86411 occurrence of such an error.

86412 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression  
86413 used in the editor; for example, for the purposes of the **s** command.

86414 If the *tagstring* is in a file with a different name than the current pathname, set the current  
86415 pathname to the name of that file, and replace the contents of the edit buffer with the contents of  
86416 that file. In this case, if no '!' is appended to the command name, and the edit buffer has been  
86417 modified since the last complete write, it shall be an error, unless the file is successfully written  
86418 as specified by the **autowrite** option.

86419 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

86420 *Current line:* If the tags file contained a line number, set to that line number. If the line number is  
86421 larger than the last line in the edit buffer, an error message shall be written and the current line  
86422 shall be set as specified for the **edit** command.

86423 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no  
86424 matching pattern is found, an error message shall be written and the current line shall be set as  
86425 specified for the **edit** command.

86426 *Current column:* If the tags file contained a line-number reference and that line-number was not  
86427 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern  
86428 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

86429 **Unabbreviate**86430 *Synopsis:*    una[bbrev] lhs86431 If *lhs* is not an entry in the current list of abbreviations (see [Abbreviate](#), on page 2654), it shall be  
86432 an error. Otherwise, delete *lhs* from the list of abbreviations.86433 *Current line:* Unchanged.86434 *Current column:* Unchanged.86435 **Undo**86436 *Synopsis:*    u[ndo]86437 Reverse the changes made by the last command that modified the contents of the edit buffer,  
86438 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands  
86439 resulting from buffer executions and mapped character expansions, are considered single  
86440 commands.86441 If no action that can be undone preceded the **undo** command, it shall be an error.86442 If the **undo** command restores lines that were marked, the mark shall also be restored unless it  
86443 was reset subsequent to the deletion of the lines.86444 *Current line:*

- 86445       1. If lines are added or changed in the file, set to the first line added or changed.
- 
- 86446       2. Set to the line before the first line deleted, if it exists.
- 
- 86447       3. Set to 1 if the edit buffer is not empty.
- 
- 86448       4. Set to zero.

86449 *Current column:* Set to non-<blank>.86450 **Unmap**86451 *Synopsis:*    unm[ap][!]*lhs*86452 If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode  
86453 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map  
86454 definitions.86455 If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command  
86456 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode  
86457 map definitions.86458 *Current line:* Unchanged.86459 *Current column:* Unchanged.

86460 **Version**86461 *Synopsis:* `ve[rsion]`86462 Write a message containing version information for the editor. The format of the message is  
86463 unspecified.86464 *Current line:* Unchanged.86465 *Current column:* Unchanged.86466 **Visual**86467 *Synopsis:* `[laddr] vi[sual][type][count][flags]`86468 If *ex* is currently in open or visual mode, the *Synopsis* and behavior of the *visual* command shall  
86469 be the same as the **edit** command, as specified by **Edit** (on page 2656).86470 Otherwise, this command need not be supported on block-mode terminals or terminals with  
86471 insufficient capabilities. If standard input, standard output, or standard error are not terminal  
86472 devices, the results are unspecified.86473 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in  
86474 **window**, on page 2683). If the '^' type character was also specified, the **window** edit option  
86475 shall be set before being used by the type character.86476 Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type*  
86477 shall cause the following effects:

86478 + Place the beginning of the specified line at the top of the display.

86479 - Place the end of the specified line at the bottom of the display.

86480 . Place the beginning of the specified line in the middle of the display.

86481 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line  
86482 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place  
86483 the beginning of this line as close to the bottom of the displayed lines as possible, while still  
86484 displaying the value of the **window** edit option number of lines.86485 *Current line:* Set to the specified line.86486 *Current column:* Set to non-<blank>.86487 **Write**86488 *Synopsis:* `[2addr] w[rite][!][>>][file]`86489 `[2addr] w[rite][!][file]`86490 `[2addr] wq[!][>>][file]`

86491 If no lines are specified, the lines shall default to the entire file.

86492 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**  
86493 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the  
86494 **quit** shall not be attempted.86495 If the command name is not followed by one or more <blank> characters, or *file* is not preceded  
86496 by a '!' character, the **write** shall be to a file.86497 1. If the >> argument is specified, and the file already exists, the lines shall be appended to  
86498 the file instead of replacing its contents. If the >> argument is specified, and the file does  
86499 not already exist, it is unspecified whether the write shall proceed as if the >> argument

- 86500 had not been specified or if the write shall fail.
- 86501 2. If the **readonly** edit option is set (see **readonly**, on page 2680), the **write** shall fail.
- 86502 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 86503 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
- 86504 the **write** command shall fail.
- 86505 5. If the current pathname is used, and the current pathname has been changed by the **file**
- 86506 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
- 86507 subsequent **writes** shall not fail for this reason (unless the current pathname is changed
- 86508 again).
- 86509 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**
- 86510 shall fail.
- 86511 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character '!' to the
- 86512 command name.
- 86513 For rules 2., 3., and 5., the **write** can be forced by setting the **writeany** edit option.
- 86514 Additional, implementation-defined tests may cause the **write** to fail.
- 86515 If the edit buffer is empty, a file without any contents shall be written.
- 86516 An informational message shall be written noting the number of lines and bytes written.
- 86517 Otherwise, if the command is followed by one or more <blank> characters, and the file is
- 86518 preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters
- 86519 expanded as described in **Command Line Parsing in ex** (on page 2646).
- 86520 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
- 86521 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a
- 86522 single argument. The specified lines shall be written to the standard input of the command. The
- 86523 standard error and standard output of the program, if any, shall be written as described for the
- 86524 **print** command. If the last character in that output is not a <newline>, a <newline> shall be
- 86525 written at the end of the output.
- 86526 The special meaning of the '!' following the **write** command can be overridden by escaping it
- 86527 with a <backslash> character.
- 86528 *Current line*: Unchanged.
- 86529 *Current column*: Unchanged.
- 86530 **Write and Exit**
- 86531 *Synopsis*: [2addr] x[it][!][file]
- 86532 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
- 86533 the **quit** command, or if a '!' is appended to the command name, to **quit!**.
- 86534 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command
- 86535 name, to **wq!**.
- 86536 *Current line*: Unchanged.
- 86537 *Current column*: Unchanged.

86538 **Yank**86539 *Synopsis:* `[2addr] ya[nk][buffer][count]`86540 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall  
86541 become a line-mode buffer.86542 *Current line:* Unchanged.86543 *Current column:* Unchanged.86544 **Adjust Window**86545 *Synopsis:* `[1addr] z[!][type ...][count][flags]`86546 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current  
86547 line value shall first be incremented by 1. If incrementing the current line would cause it to be  
86548 greater than the last line in the edit buffer, it shall be an error.86549 If there are <blank> characters between the *type* argument and the preceding *z* command name  
86550 or optional '!' character, it shall be an error.86551 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in  
86552 **window**, on page 2683). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit  
86553 option, or if ! was specified, the number of lines in the display minus 1.86554 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,  
86555 *count* lines starting with the line specified by the *type* argument shall be written.86556 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

86557 – The specified line shall be decremented by the following value:

86558  $((\text{number of ``-'' characters}) \times \text{count}) - 1$ 86559 If the calculation would result in a number less than 1, it shall be an error. Write lines from  
86560 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit  
86561 buffer has been written.

86562 + The specified line shall be incremented by the following value:

86563  $((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$ 86564 If the calculation would result in a number greater than the last line in the edit buffer, it  
86565 shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count*  
86566 lines or the last line in the edit buffer has been written.86567 =, . If more than a single ' .' or '= ' is specified, it shall be an error. The following steps shall  
86568 be taken:86569 1. If *count* is zero, nothing shall be written.86570 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count*  
86571 or '!' was specified, *N* shall be:86572  $(\text{count} - 1) / 2$ 86573 Otherwise, *N* shall be:86574  $(\text{count} - 3) / 2$ 86575 If *N* is a number less than 3, no lines shall be written.

- 86576 3. If '=' was specified as the type character, write a line consisting of the smaller of the  
86577 number of columns in the display divided by two, or 40 '-' characters.
- 86578 4. Write the current line.
- 86579 5. Repeat step 3.
- 86580 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall  
86581 be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count*  
86582 is less than 3, no lines shall be written.

86583 ^ The specified line shall be decremented by the following value:

86584  $((\text{number of } \text{'^'} \text{ characters}) + 1) \times \text{count} - 1$

86585 If the calculation would result in a number less than 1, it shall be an error. Write lines from  
86586 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit  
86587 buffer has been written.

86588 *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified  
86589 line.

86590 *Current column*: Set to non-<blank>.

### 86591 Escape

86592 *Synopsis*: ! *command*  
86593 [*addr*] ! *command*

86594 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as  
86595 described in [Command Line Parsing in ex](#) (on page 2646). If the expansion causes the text of the  
86596 line to change, it shall be redisplayed, preceded by a single '!' character.

86597 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two  
86598 arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments  
86599 to the ! command as a single argument.

86600 If no lines are specified, the standard input, standard output, and standard error of the program  
86601 shall be set to the standard input, standard output, and standard error of the *ex* program when it  
86602 was invoked. In addition, a warning message shall be written if the edit buffer has been  
86603 modified since the last complete write, and the **warn** edit option is set.

86604 If lines are specified, they shall be passed to the program as standard input, and the standard  
86605 output and standard error of the program shall replace those lines in the edit buffer. Each line in  
86606 the program output (as delimited by <newline> characters or the end of the output if it is not  
86607 immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any  
86608 occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single  
86609 <newline> characters. The specified lines shall be copied into the unnamed buffer before they  
86610 are replaced, and the unnamed buffer shall become a line-mode buffer.

86611 If in *ex* mode, a single '!' character shall be written when the program completes.

86612 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this  
86613 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this  
86614 command shall be affected by the **autoprint** edit option.

86615 *Current line:*

- 86616 1. If no lines are specified, unchanged.
- 86617 2. Otherwise, set to the last line read in, if any lines are read in.
- 86618 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 86619 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
- 86620 5. Otherwise, set to zero.

86621 *Current column:* If no lines are specified, unchanged. Otherwise, set to non-<blank>.

### 86622 **Shift Left**

86623 *Synopsis:* [2addr] <[< ...] [count] [flags]

86624 Shift the specified lines to the start of the line; the number of column positions to be shifted shall  
86625 be the number of command characters times the value of the **shiftwidth** edit option. Only  
86626 leading <blank> characters shall be deleted or changed into other <blank> characters in shifting;  
86627 other characters shall not be affected.

86628 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
86629 buffer.

86630 This command shall be affected by the **autoprint** edit option.

86631 *Current line:* Set to the last line in the lines specified.

86632 *Current column:* Set to non-<blank>.

### 86633 **Shift Right**

86634 *Synopsis:* [2addr] >[> ...] [count] [flags]

86635 Shift the specified lines away from the start of the line; the number of column positions to be  
86636 shifted shall be the number of command characters times the value of the **shiftwidth** edit  
86637 option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or  
86638 changing leading <blank> characters into other <blank> characters. Empty lines shall not be  
86639 changed.

86640 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
86641 buffer.

86642 This command shall be affected by the **autoprint** edit option.

86643 *Current line:* Set to the last line in the lines specified.

86644 *Current column:* Set to non-<blank>.

### 86645 **<control>-D**

86646 *Synopsis:* <control>-D

86647 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the  
86648 number of lines after the current line in the edit buffer. If the current line is the last line of the  
86649 edit buffer it shall be an error.

86650 *Current line:* Set to the last line written.

86651 *Current column:* Set to non-<blank>.

86652 **Write Line Number**86653 *Synopsis:* `[1addr] = [flags]`86654 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of  
86655 the specified line.86656 *Current line:* Unchanged.86657 *Current column:* Unchanged.86658 **Execute**86659 *Synopsis:* `[2addr] @ buffer`86660 `[2addr] * buffer`86661 If no buffer is specified or is specified as '@' or '\* ', the last buffer executed shall be used. If no  
86662 previous buffer has been executed, it shall be an error.86663 For each line specified by the addresses, set the current line (' ') to the specified line, and  
86664 execute the contents of the named *buffer* (as they were at the time the @ command was executed)  
86665 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode  
86666 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.86667 If an error occurs during this process, or a line specified by the addresses does not exist when  
86668 the current line would be set to it, or more than a single line was specified by the addresses, and  
86669 the contents of the edit buffer are replaced (for example, by the *ex* :edit command) an error  
86670 message shall be written, and no more commands resulting from the execution of this command  
86671 shall be processed.86672 *Current line:* As specified for the individual *ex* commands.86673 *Current column:* As specified for the individual *ex* commands.86674 **Regular Expressions in ex**86675 The *ex* utility shall support regular expressions that are a superset of the basic regular  
86676 expressions described in XBD Section 9.3 (on page 183). A null regular expression (" / ") shall  
86677 be equivalent to the last regular expression encountered.86678 Regular expressions can be used in addresses to specify lines and, in some commands (for  
86679 example, the **substitute** command), to specify portions of a line to be substituted.

86680 The following constructs can be used to enhance the basic regular expressions:

86681 \< Match the beginning of a *word*. (See the definition of *word* at the beginning of **Command**  
86682 **Descriptions in ex** (on page 2651).)86683 \> Match the end of a *word*.86684 ~ Match the replacement part of the last **substitute** command. The <tilde> (' ~ ') character can  
86685 be escaped in a regular expression to become a normal character with no special meaning.  
86686 The <backslash> shall be discarded.86687 When the editor option **magic** is not set, the only characters with special meanings shall be '^ '  
86688 at the beginning of a pattern, '\$ ' at the end of a pattern, and <backslash>. The characters '.',  
86689 '\*', '[', and '~ ' shall be treated as ordinary characters unless preceded by a <backslash>;  
86690 when preceded by a <backslash> they shall regain their special meaning, or in the case of  
86691 <backslash>, be handled as a single <backslash>. <backslash> characters used to escape other  
86692 characters shall be discarded.

86693 **Replacement Strings in ex**

86694 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall  
 86695 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not  
 86696 set) shall be replaced by the replacement part of the previous **substitute** command. The  
 86697 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the  
 86698 corresponding back-reference expression. If the corresponding back-reference expression does  
 86699 not match, then the characters '\n' shall be replaced by the empty string.

86700 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the  
 86701 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause  
 86702 the character that follows to be converted to lowercase (uppercase). The string '\L' ('\U') shall  
 86703 cause all characters subsequent to it to be converted to lowercase (uppercase) as they are  
 86704 inserted by the substitution until the string '\e' or '\E', or the end of the replacement string,  
 86705 is encountered.

86706 Otherwise, any character following a <backslash> shall be treated as that literal character, and  
 86707 the escaping <backslash> shall be discarded.

86708 An example of case conversion with the **s** command is as follows:

```
86709 :p
86710 The cat sat on the mat.
86711 :s/\<.at\>/\u&/gp
86712 The Cat Sat on the Mat.
86713 :s/S\ (. * \) M/S\U\1\eM/p
86714 The Cat SAT ON THE Mat.
```

86715 **Edit Options in ex**

86716 The *ex* utility has a number of options that modify its behavior. These options have default  
 86717 settings, which can be changed using the **set** command.

86718 Options are Boolean unless otherwise specified.

86719 **autoindent, ai**

86720 [Default *unset*]

86721 If **autoindent** is set, each line in input mode shall be indented (using first as many <tab>  
 86722 characters as possible, as determined by the editor option **tabstop**, and then using <space>  
 86723 characters) to align with another line, as follows:

- 86724 1. If in open or visual mode and the text input is part of a line-oriented command (see the  
 86725 EXTENDED DESCRIPTION in *vi*), align to the first column.
- 86726 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
  - 86727 a. If a line was previously inserted as part of this command, it shall be set to the  
 86728 indentation of the last inserted line by default, or as otherwise specified for the  
 86729 <control>-D character in **Input Mode Commands** in *vi* (on page 3344).
  - 86730 b. Otherwise, it shall be set to the indentation of the previous current line, if any;  
 86731 otherwise, to the first column.
- 86732 3. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:

- 86733 a. If a line was previously inserted as part of this command, it shall be set to the  
86734 indentation of the last inserted line by default, or as otherwise specified for the *eof*  
86735 character in **Scroll** (on page 2650).
- 86736 b. Otherwise, if the command is the *ex a* command, it shall be set to the line  
86737 appended after, if any; otherwise to the first column.
- 86738 c. Otherwise, if the command is the *ex i* command, it shall be set to the line inserted  
86739 before, if any; otherwise to the first column.
- 86740 d. Otherwise, if the command is the *ex c* command, it shall be set to the indentation of  
86741 the line replaced.

#### 86742 **autoprint, ap**

86743 [Default *set*]

86744 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the  
86745 contents of the current edit buffer, and after each **tag** command for which the tag search pattern  
86746 was found or tag line number was valid, unless:

- 86747 1. The command was executed while in open or visual mode.
- 86748 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 86749 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 86750 4. The command was the **append**, **change**, or **insert** command.
- 86751 5. The command was not terminated by a <newline>.
- 86752 6. The current line shall be written by a flag specified to the command; for example, **delete #**  
86753 shall write the current line as specified for the flag modifier to the **delete** command, and  
86754 not as specified by the **autoprint** edit option.

#### 86755 **autowrite, aw**

86756 [Default *unset*]

86757 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to  
86758 any file, the contents of the edit buffer shall be written as if the *ex write* command had been  
86759 specified without arguments, before each command affected by the **autowrite** edit option is  
86760 executed. Appending the character '!' to the command name of any of the *ex* commands  
86761 except '!' shall prevent the write. If the write fails, it shall be an error and the command shall  
86762 not be executed.

#### 86763 **beautify, bf**

86764 XSI [Default *unset*]

86765 If **beautify** is set, all non-printable characters, other than <tab>, <newline>, and <form-feed>  
86766 characters, shall be discarded from text read in from files.

- 86767           **directory, dir**  
 86768           [Default *implementation-defined*]  
 86769           The value of this option specifies the directory in which the editor buffer is to be placed. If this  
 86770           directory is not writable by the user, the editor shall quit.
- 86771           **edcompatible, ed**  
 86772           [Default *unset*]  
 86773           Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled  
 86774           by repeating the suffixes.
- 86775           **errorbells, eb**  
 86776           [Default *unset*]  
 86777           If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse  
 86778           video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.
- 86779           **exrc**  
 86780           [Default *unset*]  
 86781           If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in [Initialization in](#)  
 86782           [ex and vi](#) (on page 2642). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory  
 86783           during initialization, unless the current directory is that named by the *HOME* environment  
 86784           variable.
- 86785           **ignorecase, ic**  
 86786           [Default *unset*]  
 86787           If **ignorecase** is set, characters that have uppercase and lowercase representations shall have  
 86788           those representations considered as equivalent for purposes of regular expression comparison.  
 86789           The **ignorecase** edit option shall affect all remembered regular expressions; for example,  
 86790           unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the  
 86791           last basic regular expression in a case-sensitive fashion.
- 86792           **list**  
 86793           [Default *unset*]  
 86794           If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for  
 86795           the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall  
 86796           be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual  
 86797           text input mode, when the cursor does not rest on any character in the line, it shall rest on the  
 86798           ' \$ ' marking the end of the line.

- 86799           **magic**
- 86800           [Default *set*]
- 86801           If **magic** is set, modify the interpretation of characters in regular expressions and substitution replacement strings (see [Regular Expressions in ex](#) (on page 2675) and [Replacement Strings in ex](#), on page 2676).
- 86802
- 86803
- 86804           **mesg**
- 86805           [Default *set*]
- 86806           If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the editor started (or in a shell escape), such as:
- 86807
- 86808
- 86809
- 86810           : !mesg y
- 86811           the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable incoming messages if **mesg n** was issued.
- 86812
- 86813           **number, nu**
- 86814           [Default *unset*]
- 86815           If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line numbers, in the format specified by the **print** command with the # flag specified. In *ex* text input mode, each line shall be preceded by the line number it will have in the file.
- 86816
- 86817
- 86818           In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in the format specified by the *ex print* command with the # flag specified. This line number shall not be considered part of the line for the purposes of evaluating the current column; that is, column position 1 shall be the first column position after the format specified by the **print** command.
- 86819
- 86820
- 86821
- 86822
- 86823           **paragraphs, para**
- 86824           [Default in the POSIX locale IPLPPPQPP LIpplpipbp]
- 86825           The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual mode commands. The **paragraphs** edit option can be set to a character string consisting of zero or more character pairs. It shall be an error to set it to an odd number of characters.
- 86826
- 86827
- 86828           **prompt**
- 86829           [Default *set*]
- 86830           If **prompt** is set, *ex* command mode input shall be prompted for with a <colon> (':'); when unset, no prompt shall be written.
- 86831

86832 **readonly**86833 [Default *see text*]

86834 If the **readonly** edit option is set, read-only mode shall be enabled (see [Write](#), on page 2670). The  
 86835 **readonly** edit option shall be initialized to set if either of the following conditions are true:

- 86836 • The command-line option `-R` was specified.
- 86837 • Performing actions equivalent to the `access()` function called with the following arguments  
 86838 indicates that the file lacks write permission:
  - 86839 1. The current pathname is used as the *path* argument.
  - 86840 2. The constant `W_OK` is used as the *amode* argument.

86841 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.  
 86842 The **readonly** edit option shall not be initialized to unset based on any special privileges of the  
 86843 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the  
 86844 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly  
 86845 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again  
 86846 be reinitialized each time that the contents of the edit buffer are replaced.

86847 **redraw**86848 [Default *unset*]

86849 The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a  
 86850 large amount of output to the terminal, it is useful only at high transmission speeds.)

86851 **remap**86852 [Default *set*]

86853 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation  
 86854 shall continue until a final product is obtained. If unset, only a one-step translation shall be  
 86855 done.

86856 **report**

86857 [Default 5]

86858 The value of this **report** edit option specifies what number of lines being added, copied, deleted,  
 86859 or modified in the edit buffer will cause an informational message to be written to the user. The  
 86860 following conditions shall cause an informational message. The message shall contain the  
 86861 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- 86862 • An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the  
 86863 value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v**  
 86864 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- 86865 • An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option  
 86866 plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi*  
 86867 buffer execution, shall cause an informational message to be written.
- 86868 • An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or  
 86869 deletes a total of at least the value of the **report** edit option number of lines, and which is  
 86870 not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an  
 86871 informational message to be written. (For example, if 3 lines were added and 8 lines  
 86872 deleted during an *ex* **visual** command, 5 would be the number compared against the

- 86873 **report** edit option after the command completed.)
- 86874 **scroll, scr**
- 86875 [Default (number of lines in the display -1)/2]
- 86876 The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex*  
86877 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the  
86878 initial number of lines to scroll when no previous <control>-D or <control>-U command has  
86879 been executed.
- 86880 **sections**
- 86881 [Default in the POSIX locale `NHSHH HUnhsh`]
- 86882 The **sections** edit option shall define additional section boundaries for the open and visual mode  
86883 commands. The **sections** edit option can be set to a character string consisting of zero or more  
86884 character pairs; it shall be an error to set it to an odd number of characters.
- 86885 **shell, sh**
- 86886 [Default from the environment variable `SHELL`]
- 86887 The value of this option shall be a string. The default shall be taken from the `SHELL`  
86888 environment variable. If the `SHELL` environment variable is null or empty, the *sh* (see *sh*) utility  
86889 shall be the default.
- 86890 **shiftwidth, sw**
- 86891 [Default 8]
- 86892 The value of this option shall give the width in columns of an indentation level used during  
86893 autoindentation and by the shift commands (< and >).
- 86894 **showmatch, sm**
- 86895 [Default *unset*]
- 86896 The functionality described for the **showmatch** edit option need not be supported on block-  
86897 mode terminals or terminals with insufficient capabilities.
- 86898 If **showmatch** is set, in open or visual mode, when a ' ) ' or ' } ' is typed, if the matching ' ( ' or  
86899 ' { ' is currently visible on the display, the matching ' ( ' or ' { ' shall be flagged moving the  
86900 cursor to its location for an unspecified amount of time.
- 86901 **showmode**
- 86902 [Default *unset*]
- 86903 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be  
86904 displayed on the last line of the display. Command mode and text input mode shall be  
86905 differentiated; other unspecified modes and implementation-defined information may be  
86906 displayed.

- 86907           **slowopen**
- 86908           [Default *unset*]
- 86909           If **slowopen** is set during open and visual text input modes, the editor shall not update portions  
86910           of the display other than those display line columns that display the characters entered by the  
86911           user (see [Input Mode Commands in vi](#), on page 3344).
- 86912           **tabstop, ts**
- 86913           [Default 8]
- 86914           The value of this edit option shall specify the column boundary used by a <tab> in the display  
86915           (see [autoprint, ap](#) (on page 2677) and [Input Mode Commands in vi](#), on page 3344).
- 86916           **taglength, tl**
- 86917           [Default zero]
- 86918           The value of this edit option shall specify the maximum number of characters that are  
86919           considered significant in the user-specified tag name and in the tag name from the tags file. If  
86920           the value is zero, all characters in both tag names shall be significant.
- 86921           **tags**
- 86922           [Default *see text*]
- 86923           The value of this edit option shall be a string of <blank>-delimited pathnames of files used by  
86924           the **tag** command. The default value is unspecified.
- 86925           **term**
- 86926           [Default from the environment variable *TERM*]
- 86927           The value of this edit option shall be a string. The default shall be taken from the *TERM* variable  
86928           in the environment. If the *TERM* environment variable is empty or null, the default is  
86929           unspecified. The editor shall use the value of this edit option to determine the type of the display  
86930           device.
- 86931           The results are unspecified if the user changes the value of the term edit option after editor  
86932           initialization.
- 86933           **terse**
- 86934           [Default *unset*]
- 86935           If **terse** is set, error messages may be less verbose. However, except for this caveat, error  
86936           messages are unspecified. Furthermore, not all error messages need change for different settings  
86937           of this option.

- 86938           **warn**
- 86939           [Default *set*]
- 86940           If **warn** is set, and the contents of the edit buffer have been modified since they were last  
86941 completely written, the editor shall write a warning message before certain ! commands (see  
86942 [Escape](#), on page 2673).
- 86943           **window**
- 86944           [Default *see text*]
- 86945           A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in  
86946 visual mode, to specify the number of lines displayed when the screen is repainted.
- 86947           If the **-w** command-line option is not specified, the default value shall be set to the value of the  
86948 *LINES* environment variable. If the *LINES* environment variable is empty or null, the default  
86949 shall be the number of lines in the display minus 1.
- 86950           Setting the **window** edit option to zero or to a value greater than the number of lines in the  
86951 display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)  
86952 shall cause the **window** edit option to be set to the number of lines in the display minus 1.
- 86953           The baud rate of the terminal line may change the default in an implementation-defined manner.
- 86954           **wrapmargin, wm**
- 86955           [Default 0]
- 86956           If the value of this edit option is zero, it shall have no effect.
- 86957           If not in the POSIX locale, the effect of this edit option is implementation-defined.
- 86958           Otherwise, it shall specify a number of columns from the ending margin of the terminal.
- 86959           During open and visual text input modes, for each character for which any part of the character  
86960 is displayed in a column that is less than **wrapmargin** columns from the ending margin of the  
86961 display line, the editor shall behave as follows:
- 86962           1. If the character triggering this event is a <blank>, it, and all immediately preceding  
86963 <blank> characters on the current line entered during the execution of the current text  
86964 input command, shall be discarded, and the editor shall behave as if the user had entered  
86965 a single <newline> instead. In addition, if the next user-entered character is a <space>, it  
86966 shall be discarded as well.
- 86967           2. Otherwise, if there are one or more <blank> characters on the current line immediately  
86968 preceding the last group of inserted non-<blank> characters which was entered during  
86969 the execution of the current text input command, the <blank> characters shall be replaced  
86970 as if the user had entered a single <newline> instead.
- 86971           If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any  
86972 <blank> characters at or after the cursor in the current line shall be discarded.
- 86973           The ending margin shall be determined by the system or overridden by the user, as described for  
86974 *COLUMNS* in the ENVIRONMENT VARIABLES section and XBD [Chapter 8](#) (on page 173).

- 86975           **wrapscan, ws**
- 86976           [Default *set*]
- 86977           If **wrapscan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and n commands) shall wrap around the beginning or end of the edit buffer; when unset, searches shall stop at the beginning or end of the edit buffer.
- 86978
- 86979
- 86980           **writeany, wa**
- 86981           [Default *unset*]
- 86982           If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be inhibited, as described in editor option **autowrite**.
- 86983
- 86984   **EXIT STATUS**
- 86985           The following exit values shall be returned:
- 86986           0   Successful completion.
- 86987           >0   An error occurred.
- 86988   **CONSEQUENCES OF ERRORS**
- 86989           When any error is encountered and the standard input is not a terminal device file, *ex* shall not write the file or return to command or text input mode, and shall terminate with a non-zero exit status.
- 86990
- 86991
- 86992           Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP asynchronous event.
- 86993
- 86994           Otherwise, when an error is encountered, the editor shall behave as specified in [Command Line Parsing in ex](#) (on page 2646).
- 86995
- 86996   **APPLICATION USAGE**
- 86997           If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.
- 86998           The **next** command can accept more than one file, so usage such as:
- 86999           next `ls [abc]\*`
- 87000           is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect only one file and unspecified results occur.
- 87001
- 87002   **EXAMPLES**
- 87003           None.
- 87004   **RATIONALE**
- 87005           The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V implementations of *ex* and *vi*.
- 87006
- 87007           A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and rejected for inclusion. Neither option provided the level of security that users might expect.
- 87008
- 87009           It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing; thus, it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that an *ex* implementation should provide the full set of capabilities on all terminals capable of supporting them.
- 87010
- 87011
- 87012
- 87013

87014 **Options**

87015 The **-c** replacement for **+command** was inspired by the **-e** option of *sed*. Historically, all such  
 87016 commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This  
 87017 meant, for example, that **"/pattern"** would fail unless the **wrapsan** option was set.  
 87018 POSIX.1-2008 requires conformance to historical practice. The **+command** option is no longer  
 87019 specified by POSIX.1-2008 but may be present in some implementations. Historically, some  
 87020 implementations restricted the *ex* commands that could be listed as part of the command line  
 87021 arguments. For consistency, POSIX.1-2008 does not permit these restrictions.

87022 In historical implementations of the editor, the **-R** option (and the **readonly** edit option) only  
 87023 prevented overwriting of files; appending to files was still permitted, mapping loosely into the  
 87024 *cs*h **noclobber** variable. Some implementations, however, have not followed this semantic, and  
 87025 **readonly** does not permit appending either. POSIX.1-2008 follows the latter practice, believing  
 87026 that it is a more obvious and intuitive meaning of **readonly**.

87027 The **-s** option suppresses all interactive user feedback and is useful for editing scripts in batch  
 87028 jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting  
 87029 open and visual modes" has historically been named "dumb".

87030 The **-t** option was required because the *ctags* utility appears in POSIX.1-2008 and the option is  
 87031 available in all historical implementations of *ex*.

87032 Historically, the *ex* and *vi* utilities accepted a **-x** option, which did encryption based on the  
 87033 algorithm found in the historical *crypt* utility. The **-x** option for encryption, and the associated  
 87034 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control  
 87035 laws of some nations make it difficult to export cryptographic technology. In addition, it did not  
 87036 historically provide the level of security that users might expect.

87037 **Standard Input**

87038 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file  
 87039 character, <control>-D, is historically an *ex* command.

87040 There was no maximum line length in historical implementations of *ex*. Specifically, as it was  
 87041 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the  
 87042 maximum line buffer size was declared as BUFSIZ, which was different lengths on different  
 87043 systems. This version selected the value of {LINE\_MAX} to impose a reasonable restriction on  
 87044 portable usage of *ex* and to aid test suite writers in their development of realistic tests that  
 87045 exercise this limit.

87046 **Input Files**

87047 It was an explicit decision by the standard developers that a <newline> be added to any file  
 87048 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make  
 87049 text files lacking a trailing <newline> more portable. It is recognized that this will require a user-  
 87050 specified option or extension for implementations that permit *ex* and *vi* to edit files of type other  
 87051 than text if such files are not otherwise identified by the system. It was agreed that the ability to  
 87052 edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an  
 87053 *ex* or *vi* implementation be required to handle files other than text files.

87054 The paragraph in the INPUT FILES section, "By default, ...", is intended to close a long-  
 87055 standing security problem in *ex* and *vi*; that of the "modeline" or "modelines" edit option. This  
 87056 feature allows any line in the first or last five lines of the file containing the strings "ex:" or  
 87057 "vi:" (and, apparently, "ei:" or "vx:") to be a line containing editor commands, and *ex*  
 87058 interprets all the text up to the next ':' or <newline> as a command. Consider the

87059 consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to  
87060 a mail message in which a line such as:

87061 `ex:! rm -rf :`

87062 appeared in the signature lines. The standard developers believed strongly that an editor should  
87063 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from  
87064 their implementations of *ex* and *vi*.

### 87065 **Asynchronous Events**

87066 The intention of the phrase “complete write” is that the entire edit buffer be written to stable  
87067 storage. The note regarding temporary files is intended for implementations that use temporary  
87068 files to back edit buffers unnamed by the user.

87069 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the Q command in visual  
87070 mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-2008 permits, but does not  
87071 require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input  
87072 mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the  
87073 terminal on this event, and some did not. POSIX.1-2008 requires that SIGINT behave identically  
87074 to <ESC>, and that the terminal not be alerted.

87075 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as  
87076 completed lines were retained, but any partial line discarded, and the editor returned to  
87077 command mode. POSIX.1-2008 is silent on this issue; implementations are encouraged to follow  
87078 historical practice, where possible.

87079 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore  
87080 impossible to suspend the editor in visual text input mode. There are two major reasons for this.  
87081 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where  
87082 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if  
87083 SIGTSTP was delivered to the process group in the default manner. The second was that most  
87084 implementations of the UNIX *curses* package did not handle SIGTSTP safely, and the receipt of  
87085 SIGTSTP at the wrong time would cause them to crash. POSIX.1-2008 is silent on this issue;  
87086 implementations are encouraged to treat suspension as an asynchronous event if possible.

87087 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were  
87088 retained; that is, anywhere from zero to all of the lines to be modified might have been modified  
87089 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.  
87090 POSIX.1-2008 permits this behavior, noting that the **undo** command is required to be able to  
87091 undo these partially completed commands.

87092 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is  
87093 unspecified because some implementations attempt to save the edit buffer in a useful state when  
87094 other signals are received.

### 87095 **Standard Error**

87096 For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to  
87097 invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination  
87098 condition. Diagnostic messages should not be confused with the error messages generated by  
87099 inappropriate or illegal user commands.

87100 **Initialization in ex and vi**

87101 If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the  
87102 alternate and current pathnames will be set. Informally, they are set as follows:

- 87103 1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the  
87104 current pathname will be set to the filename argument (the first filename argument in the  
87105 case of the **next** command) and the alternate pathname will be set to the previous current  
87106 pathname, if there was one.
- 87107 2. In the case of the file read/write forms of the **read** and **write** commands, if there is no  
87108 current pathname, the current pathname will be set to the filename argument.
- 87109 3. Otherwise, the alternate pathname will be set to the filename argument.

87110 For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there  
87111 was a previous current pathname, the alternate pathname. The commands **:write**, **!command**,  
87112 and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for  
87113 some reason, the alternate pathname would be set. The **read** and **write** commands set the  
87114 alternate pathname to their *file* argument, unless the current pathname is not set, in which case  
87115 they set the current pathname to their *file* arguments. The alternate pathname was not  
87116 historically set by the **:source** command. POSIX.1-2008 requires conformance to historical  
87117 practice. Implementations adding commands that take filenames as arguments are encouraged  
87118 to set the alternate pathname as described here.

87119 Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed  
87120 in the *\$HOME* directory. POSIX.1-2008 prohibits this behavior.

87121 Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the  
87122 real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was  
87123 a security problem because it is possible to put normal UNIX system commands inside a **.exrc**  
87124 file. POSIX.1-2008 does not specify the **sourceany** option, and historical implementations are  
87125 encouraged to delete it.

87126 The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than  
87127 the owner. The appropriate privileges exception is intended to permit users to acquire special  
87128 privileges, but continue to use the **.exrc** files in their home directories.

87129 System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is  
87130 that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off,  
87131 so by default local **.exrc** files were not read. The problem this was intended to solve was that  
87132 System V permitted users to give away files, so there is no possible ownership or writeability  
87133 test to ensure that the file is safe. This is still a security problem on systems where users can give  
87134 away files, but there is nothing additional that POSIX.1-2008 can do. The implementation-  
87135 defined exception is intended to permit groups to have local **.exrc** files that are shared by users,  
87136 by creating pseudo-users to own the shared files.

87137 POSIX.1-2008 does not mention system-wide *ex* and *vi* start-up files. While they exist in several  
87138 implementations of *ex* and *vi*, they are not present in any implementations considered historical  
87139 practice by POSIX.1-2008. Implementations that have such files should use them only if they are  
87140 owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if  
87141 they are not writable by any user other than their owner. System-wide start-up files should be  
87142 read before the *EXINIT* variable, *\$HOME/.exrc*, or local **.exrc** files are evaluated.

87143 Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although  
87144 ones requiring that the edit buffer already contain lines of text generally caused historical  
87145 implementations of the editor to drop **core**. POSIX.1-2008 requires that any *ex* command be

87146 permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and consistency,  
 87147 although many of them will obviously fail under many circumstances.

87148 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with  
 87149 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded  
 87150 during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc*  
 87151 file read in the contents of a file and did not subsequently write the edit buffer. An additional  
 87152 intent of this phrase is to specify that the initial current line and column is set as specified for the  
 87153 individual *ex* commands.

87154 Historically, the *-t* option behaved as if the tag search were a *+command*; that is, it was executed  
 87155 from the last line of the file specified by the tag. This resulted in the search failing if the pattern  
 87156 was a forward search pattern and the *wrapsan* edit option was not set. POSIX.1-2008 does not  
 87157 permit this behavior, requiring that the search for the tag pattern be performed on the entire file,  
 87158 and, if not found, that the current line be set to a more reasonable location in the file.

87159 Historically, the empty edit buffer presented for editing when a file was not specified by the user  
 87160 was unnamed. This is permitted by POSIX.1-2008; however, implementations are encouraged to  
 87161 provide users a temporary filename for this buffer because it permits them the use of *ex*  
 87162 commands that use the current pathname during temporary edit sessions.

87163 Historically, the file specified using the *-t* option was not part of the current argument list. This  
 87164 practice is permitted by POSIX.1-2008; however, implementations are encouraged to include its  
 87165 name in the current argument list for consistency.

87166 Historically, the *-c* command was generally not executed until a file that already exists was  
 87167 edited. POSIX.1-2008 requires conformance to this historical practice. Commands that could  
 87168 cause the *-c* command to be executed include the *ex* commands *edit*, *next*, *recover*, *rewind*, and  
 87169 *tag*, and the *vi* commands *<control>-^* and *<control>-]*. Historically, reading a file into an edit  
 87170 buffer did not cause the *-c* command to be executed (even though it might set the current  
 87171 pathname) with the exception that it did cause the *-c* command to be executed if: the editor was  
 87172 in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read  
 87173 commands had yet been attempted. For consistency and simplicity of specification,  
 87174 POSIX.1-2008 does not permit this behavior.

87175 Historically, the *-r* option was the same as a normal edit session if there was no recovery  
 87176 information available for the file. This allowed users to enter:

87177 `vi -r *.c`

87178 and recover whatever files were recoverable. In some implementations, recovery was attempted  
 87179 only on the first file named, and the file was not entered into the argument list; in others,  
 87180 recovery was attempted for each file named. In addition, some historical implementations  
 87181 ignored *-r* if *-t* was specified or did not support command line *file* arguments with the *-t* option.  
 87182 For consistency and simplicity of specification, POSIX.1-2008 disallows these special cases, and  
 87183 requires that recovery be attempted the first time each file is edited.

87184 Historically, *vi* initialized the *'* and *'* marks, but *ex* did not. This meant that if the first command  
 87185 in *ex* mode was *visual* or if an *ex* command was executed first (for example, *vi +10 file*), *vi*  
 87186 was entered without the marks being initialized. Because the standard developers believed the marks  
 87187 to be generally useful, and for consistency and simplicity of specification, POSIX.1-2008 requires  
 87188 that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is  
 87189 not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however,  
 87190 it has always been possible to set (and use) marks in empty edit buffers in open and visual mode  
 87191 edit sessions.

87192 **Addressing**

87193 Historically, *ex* and *vi* accepted the additional addressing forms '`\/'` and '`\?'`. They were  
87194 equivalent to "`//`" and "`??`", respectively. They are not required by POSIX.1-2008, mostly  
87195 because nobody can remember whether they ever did anything different historically.

87196 Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the %  
87197 address in empty files for others. For consistency, POSIX.1-2008 requires support for the former  
87198 in the few commands where it makes sense, and disallows it otherwise. In addition, because  
87199 POSIX.1-2008 requires that % be logically equivalent to "`1, $`", it is also supported where it  
87200 makes sense and disallowed otherwise.

87201 Historically, the % address could not be followed by further addresses. For consistency and  
87202 simplicity of specification, POSIX.1-2008 requires that additional addresses be supported.

87203 All of the following are valid *addresses*:

|       |                       |                                                    |
|-------|-----------------------|----------------------------------------------------|
| 87204 | <code>+++</code>      | Three lines after the current line.                |
| 87205 | <code>/re/-</code>    | One line before the next occurrence of <i>re</i> . |
| 87206 | <code>-2</code>       | Two lines before the current line.                 |
| 87207 | <code>3 ---- 2</code> | Line one (note intermediate negative address).     |
| 87208 | <code>1 2 3</code>    | Line six.                                          |

87209 Any number of addresses can be provided to commands taking addresses; for example,  
87210 "`1, 2, 3, 4, 5p`" prints lines 4 and 5, because two is the greatest valid number of addresses  
87211 accepted by the **print** command. This, in combination with the <semicolon> delimiter, permits  
87212 users to create commands based on ordered patterns in the file. For example, the command  
87213 **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next  
87214 two lines. Note that the address **3;** must be evaluated before being discarded because the search  
87215 origin for the **/foo/** command depends on this.

87216 Historically, values could be added to addresses by including them after one or more <blank>  
87217 characters; for example, `3 - 5p` wrote the seventh line of the file, and **/foo/ 5** was the same as  
87218 **/foo/+5**. However, only absolute values could be added; for example, `5 /foo/` was an error.  
87219 POSIX.1-2008 requires conformance to historical practice. Address offsets are separately  
87220 specified from addresses because they could historically be provided to visual mode search  
87221 commands.

87222 Historically, any missing addresses defaulted to the current line. This was true for leading and  
87223 trailing <comma>-delimited addresses, and for trailing <semicolon>-delimited addresses. For  
87224 consistency, POSIX.1-2008 requires it for leading <semicolon> addresses as well.

87225 Historically, *ex* and *vi* accepted the '`^`' character as both an address and as a flag offset for  
87226 commands. In both cases it was identical to the '`-`' character. POSIX.1-2008 does not require or  
87227 prohibit this behavior.

87228 Historically, the enhancements to basic regular expressions could be used in addressing; for  
87229 example, '`~`', '`\<`', and '`\>`'. POSIX.1-2008 requires conformance to historical practice; that  
87230 is, that regular expression usage be consistent, and that regular expression enhancements be  
87231 supported wherever regular expressions are used.

87232 **Command Line Parsing in ex**

87233 Historical *ex* command parsing was even more complex than that described here. POSIX.1-2008  
 87234 requires the subset of the command parsing that the standard developers believed was  
 87235 documented and that users could reasonably be expected to use in a portable fashion, and that  
 87236 was historically consistent between implementations. (The discarded functionality is obscure, at  
 87237 best.) Historical implementations will require changes in order to comply with POSIX.1-2008;  
 87238 however, users are not expected to notice any of these changes. Most of the complexity in *ex*  
 87239 parsing is to handle three special termination cases:

- 87240 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by  
 87241 <newline> characters (they can contain <vertical-line> characters that are usually shell  
 87242 pipes).
- 87243 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,  
 87244 optionally containing <vertical-line> characters, as their first arguments.
- 87245 3. The **s** command takes a regular expression as its first argument, and uses the delimiting  
 87246 characters to delimit the command.

87247 Historically, <vertical-line> characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and  
 87248 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the  
 87249 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did  
 87250 not delimit the command at all. For example, the following commands are all valid:

```
87251 :edit +25 | s/abc/ABC/ file.c
87252 :s/ | /PIPE/
87253 :read !spell % | columnate
87254 :global/pattern/p | l
87255 :s/a/b/ | s/c/d | set
```

87256 Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT*  
 87257 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.  
 87258 POSIX.1-2008 specifically requires that they be ignored when encountered in **.exrc** and **sourced**  
 87259 files to eliminate a common source of new user error.

87260 Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were  
 87261 handled oddly when executed from *ex* mode. For example, the command `||| <carriage-return>`,  
 87262 when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|`  
 87263 would only display the line after the next line, instead of the next two lines. The former worked  
 87264 more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-2008  
 87265 requires the *vi* behavior; that is, a single default command and line number increment for each  
 87266 command separator, and trailing <newline> characters after <vertical-line> separators are  
 87267 discarded.

87268 Historically, *ex* permitted a single extra <colon> as a leading command character; for example,  
 87269 **:g/pattern/:p** was a valid command. POSIX.1-2008 generalizes this to require that any number of  
 87270 leading <colon> characters be stripped.

87271 Historically, any prefix of the **delete** command could be followed without intervening <blank>  
 87272 characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*.  
 87273 POSIX.1-2008 requires conformance to historical practice.

87274 Historically, the **k** command could be followed by the mark name without intervening <blank>  
 87275 characters. POSIX.1-2008 requires conformance to historical practice.

87276 Historically, the **s** command could be immediately followed by flag and option characters; for  
 87277 example, **s/e/E/l s |sgc3p** was a valid command. However, flag characters could not stand alone;

87278 for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would  
 87279 succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the  
 87280 command.) Another issue was that option characters had to precede flag characters even when  
 87281 the command was fully specified; for example, the command **s/e/E/pg** would fail, while the  
 87282 command **s/e/E/gp** would succeed. POSIX.1-2008 requires conformance to historical practice.

87283 Historically, the first command name that had a prefix matching the input from the user was the  
 87284 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.  
 87285 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.  
 87286 POSIX.1-2008 requires conformance to historical practice. The restriction on command search  
 87287 order for implementations with extensions is to avoid the addition of commands such that the  
 87288 historical prefixes would fail to work portably.

87289 Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands,  
 87290 separated by <vertical-line> characters, that entered or exited visual mode or the editor. Because  
 87291 implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-2008 does not permit  
 87292 it.

87293 The requirement that alphabetic command names consist of all following alphabetic characters  
 87294 up to the next non-alphabetic character means that alphabetic command names must be  
 87295 separated from their arguments by one or more non-alphabetic characters, normally a <blank>  
 87296 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

87297 Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*,  
 87298 <newline>, <carriage-return>) erased any prompting character and displayed the next lines  
 87299 without scrolling the terminal; that is, immediately below any previously displayed lines. This  
 87300 provided a cleaner presentation of the lines in the file for the user. POSIX.1-2008 does not require  
 87301 this behavior because it may be impossible in some situations; however, implementations are  
 87302 strongly encouraged to provide this semantic if possible.

87303 Historically, it was possible to change files in the middle of a command, and have the rest of the  
 87304 command executed in the new file; for example:

```
87305 :edit +25 file.c | s/abc/ABC/ | 1
```

87306 was a valid command, and the substitution was attempted in the newly edited file.  
 87307 POSIX.1-2008 requires conformance to historical practice. The following commands are  
 87308 examples that exercise the *ex* parser:

```
87309 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
87310 vi
```

```
87311 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

87312 Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or **\***  
 87313 commands changing edit buffers during execution of their associated commands. Because this  
 87314 would almost invariably result in catastrophic failure of the editor, and implementations exist  
 87315 that do exhibit these problems, POSIX.1-2008 requires that changing the edit buffer during a  
 87316 **global** or **v** command, or during a **@** or **\*** command for which there will be more than a single  
 87317 execution, be an error. Implementations supporting multiple edit buffers simultaneously are  
 87318 strongly encouraged to apply the same semantics to switching between buffers as well.

87319 The *ex* command quoting required by POSIX.1-2008 is a superset of the quoting in historical  
 87320 implementations of the editor. For example, it was not historically possible to escape a <blank>  
 87321 in a filename; for example, **:edit foo\\ bar** would report that too many filenames had been  
 87322 entered for the edit command, and there was no method of escaping a <blank> in the first  
 87323 argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-2008 extends historical  
 87324 practice, requiring that quoting behavior be made consistent across all *ex* commands, except for

87325 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V  
87326 instead of <backslash> characters for quoting. For those four commands, POSIX.1-2008 requires  
87327 conformance to historical practice.

87328 Backslash quoting in *ex* is non-intuitive. <backslash>-escapes are ignored unless they escape a  
87329 special character; for example, when performing *file* argument expansion, the string "\\%" is  
87330 equivalent to '\%', not "\<current\_pathname>". This can be confusing for users because  
87331 <backslash> is usually one of the characters that causes shell expansion to be performed, and  
87332 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are  
87333 only considered if they escape a special character, and a quoting character must be provided for  
87334 each layer of parsing for which the character is special. As another example, only a single  
87335 <backslash> is necessary for the '\1' sequence in substitute replacement patterns, because the  
87336 character '1' is not special to any parsing layer above it.

87337 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands  
87338 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character  
87339 may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-2008  
87340 requires conformance to historical practice.

87341 Historical implementations of the editor did not require delimiters within character classes to be  
87342 escaped; for example, the command **s[/][/]** on the string "xxx/yyy" would delete the '/' from  
87343 the string. POSIX.1-2008 disallows this historical practice for consistency and because it places a  
87344 large burden on implementations by requiring that knowledge of regular expressions be built  
87345 into the editor parser.

87346 Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most  
87347 cases, the <newline> character always terminated the command, regardless of any preceding  
87348 escape character, because <backslash> characters did not escape <newline> characters for most  
87349 *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted  
87350 <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V  
87351 characters escaped them instead of <backslash> characters). This was true in not only the  
87352 command line, but also **.exrc** and **sourced** files. For example, the command:

```
87353 map = foo<control-V><newline>bar
```

87354 would succeed, although it was sometimes difficult to get the <control>-V and the inserted  
87355 <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-2008  
87356 requires that it be possible to escape <newline> characters in *ex* commands at all times, using  
87357 <backslash> characters for most *ex* commands, and using <control>-V characters for the **map**  
87358 and **abbreviation** commands. For example, the command **print<newline>list** is required to be  
87359 parsed as the single command **print<newline>list**. While this differs from historical practice,  
87360 POSIX.1-2008 developers believed it unlikely that any script or user depended on the historical  
87361 behavior.

87362 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**  
87363 commands to be discarded. POSIX.1-2008 disallows this for consistency with mapped keys, the  
87364 **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

87365 **Input Editing in *ex***

87366 One of the common uses of the historical *ex* editor is over slow network connections. Editors that  
 87367 run in canonical mode can require far less traffic to and from, and far less processing on, the host  
 87368 machine, as well as more easily supporting block-mode terminals. For these reasons,  
 87369 POSIX.1-2008 requires that *ex* be implemented using canonical mode input processing, as was  
 87370 done historically.

87371 POSIX.1-2008 does not require the historical 4 BSD input editing characters “word erase” or  
 87372 “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they must  
 87373 have the required effect. Implementations that resolve them after the line has been ended using a  
 87374 <newline> or <control>-M character, and implementations that rely on the underlying system  
 87375 terminal support for this processing, are both conforming. Implementations are strongly urged  
 87376 to use the underlying system functionality, if at all possible, for compatibility with other system  
 87377 text input interfaces.

87378 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor  
 87379 moved to display the new end of the **autoindent** characters, but did not move the cursor to a  
 87380 new line, nor did it erase the <control>-D character from the line. POSIX.1-2008 does not specify  
 87381 that the cursor remain on the same line or that the rest of the line is erased; however,  
 87382 implementations are strongly encouraged to provide the best possible user interface; that is, the  
 87383 cursor should remain on the same line, and any <control>-D character on the line should be  
 87384 erased.

87385 POSIX.1-2008 does not require the historical 4 BSD input editing character “reprint”,  
 87386 traditionally <control>-R, which redisplayed the current input from the user. For this reason,  
 87387 and because the functionality cannot be implemented after the line has been terminated by the  
 87388 user, POSIX.1-2008 makes no requirements about this functionality. Implementations are  
 87389 strongly urged to make this historical functionality available, if possible.

87390 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.  
 87391 POSIX.1-2008 requires conformance to historical practice to avoid breaking historical *ex* scripts  
 87392 and **.exrc** files.

87393 **eof**

87394 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left  
 87395 unspecified so that implementations can conform in the presence of systems that do not support  
 87396 this functionality. Implementations are encouraged to modify the line and redisplay it  
 87397 immediately, if possible.

87398 The specification of the handling of the *eof* character differs from historical practice only in that  
 87399 *eof* characters are not discarded if they follow normal characters in the text input. Historically,  
 87400 they were always discarded.

87401 **Command Descriptions in *ex***

87402 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and  
 87403 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit  
 87404 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or  
 87405 command execution in an empty file, make sense only for commands that add new text to the  
 87406 edit buffer or write commands (because users may wish to write empty files). POSIX.1-2008  
 87407 requires this behavior for such commands and disallows it otherwise, for consistency and  
 87408 simplicity of specification.

87409 A count to an *ex* command has been historically corrected to be no greater than the last line in a

87410 file; for example, in a five-line file, the command **1,6print** would fail, but the command  
87411 **1print300** would succeed. POSIX.1-2008 requires conformance to historical practice.

87412 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as  
87413 described by POSIX.1-2008, but there were some special cases. For instance, the **list**, **number**,  
87414 and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line  
87415 3, and 3 would be the current line after the execution of the command. The **open** and **visual**  
87416 commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the  
87417 **open** and **visual** commands interacted badly with the **list** edit option, and setting and then  
87418 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the  
87419 specified format. For consistency and simplicity of specification, POSIX.1-2008 does not permit  
87420 any of these exceptions to the general rule.

87421 POSIX.1-2008 uses the word *copy* in several places when discussing buffers. This is not intended  
87422 to imply implementation.

87423 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would  
87424 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a  
87425 *count*. POSIX.1-2008 requires conformance to historical practice by default, but does not  
87426 preclude extensions.

87427 Historically, the contents of the unnamed buffer were frequently discarded after commands that  
87428 did not explicitly affect it; for example, when using the **edit** command to switch files. For  
87429 consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior.

87430 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting  
87431 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user  
87432 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric  
87433 buffers would not have changed. POSIX.1-2008 requires conformance to historical practice.  
87434 Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a  
87435 single location in POSIX.1-2008.

87436 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as  
87437 does the method for doing shell expansion. Implementations wishing to provide users with the  
87438 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit  
87439 option.

87440 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to  
87441 do so; for example, **!date > /dev/null** does not require a screen refresh because the output of  
87442 the UNIX *date* command requires only a single line of the screen. POSIX.1-2008 requires that the  
87443 screen be refreshed if it has been overwritten, but makes no requirements as to how an  
87444 implementation should make that determination. Implementations may prompt and refresh the  
87445 screen regardless.

#### 87446 **Abbreviate**

87447 Historical practice was that characters that were entered as part of an abbreviation replacement  
87448 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,  
87449 and so on; that is, they were logically pushed onto the terminal input queue, and were not a  
87450 simple replacement. POSIX.1-2008 requires conformance to historical practice. Historical  
87451 practice was that whenever a non-word character (that had not been escaped by a <control>-V)  
87452 was entered after a word character, *vi* would check for abbreviations. The check was based on  
87453 the type of the character entered before the word character of the word/non-word pair that  
87454 triggered the check. The word character of the word/non-word pair that triggered the check and  
87455 all characters entered before the trigger pair that were of that type were included in the check,  
87456 with the exception of <blank> characters, which always delimited the abbreviation.

87457 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can  
 87458 be no transitions from word to non-word characters (or *vice versa*) other than between the last  
 87459 and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In  
 87460 addition, because of the historical quoting rules, it was impossible to enter a literal <control>-V  
 87461 in the *lhs*. POSIX.1-2008 requires conformance to historical practice. Historical implementations  
 87462 did not inform users when abbreviations that could never be used were entered;  
 87463 implementations are strongly encouraged to do so.

87464 For example, the following abbreviations will work:

```
87465 :ab (p REPLACE
87466 :ab p REPLACE
87467 :ab ( (p REPLACE
```

87468 The following abbreviations will not work:

```
87469 :ab ( REPLACE
87470 :ab (pp REPLACE
```

87471 Historical practice is that words on the *vi* colon command line were subject to abbreviation  
 87472 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**  
 87473 command. Because there are implementations that do not do abbreviation expansion for the first  
 87474 argument to those commands, this is permitted, but not required, by POSIX.1-2008. However,  
 87475 the following sequence:

```
87476 :ab foo bar
87477 :ab foo baz
```

87478 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and  
 87479 the sequence:

```
87480 :ab fool bar
87481 :ab foo2 bar
87482 :unabbreviate foo2
```

87483 deleted the abbreviation "fool", not "foo2". These behaviors are not permitted by  
 87484 POSIX.1-2008 because they clearly violate the expectations of the user.

87485 It was historical practice that <control>-V, not <backslash>, characters be interpreted as escaping  
 87486 subsequent characters in the **abbreviate** command. POSIX.1-2008 requires conformance to  
 87487 historical practice; however, it should be noted that an abbreviation containing a <blank> will  
 87488 never work.

## 87489 Append

87490 Historically, any text following a <vertical-line> command separator after an **append**, **change**, or  
 87491 **insert** command became part of the insert text. For example, in the command:

```
87492 :g/pattern/append|stuff1
```

87493 a line containing the text "stuff1" would be appended to each line matching pattern. It was  
 87494 also historically valid to enter:

```
87495 :append|stuff1
87496 stuff2
87497 .
```

87498 and the text on the *ex* command line would be appended along with the text inserted after it.  
 87499 There was an historical bug, however, that the user had to enter two terminating lines (the ' . '

87500 lines) to terminate text input mode in this case. POSIX.1-2008 requires conformance to historical  
87501 practice, but disallows the historical need for multiple terminating lines.

### 87502 **Change**

87503 See the RATIONALE for the **append** command. Historical practice for cursor positioning after  
87504 the change command when no text is input, is as described in POSIX.1-2008. However, one  
87505 System V implementation is known to have been modified such that the cursor is positioned on  
87506 the first address specified, and not on the line before the first address. POSIX.1-2008 disallows  
87507 this modification for consistency.

87508 Historically, the **change** command did not support buffer arguments, although some  
87509 implementations allow the specification of an optional buffer. This behavior is neither required  
87510 nor disallowed by POSIX.1-2008.

### 87511 **Change Directory**

87512 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as  
87513 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have  
87514 `'.'` or `".."` as their first component. Elements in the **cdpath** edit option are `<colon>`-separated.  
87515 The initial value of the **cdpath** edit option is the value of the shell `CDPATH` environment  
87516 variable. This feature was not included in POSIX.1-2008 because it does not exist in any of the  
87517 implementations considered historical practice.

### 87518 **Copy**

87519 Historical implementations of *ex* permitted copies to lines inside of the specified range; for  
87520 example, **:2,5copy3** was a valid command. POSIX.1-2008 requires conformance to historical  
87521 practice.

### 87522 **Delete**

87523 POSIX.1-2008 requires support for the historical parsing of a **delete** command followed by flags,  
87524 without any intervening `<blank>` characters. For example:

87525 **1dp** Deletes the first line and prints the line that was second.

87526 **1delep** As for **1dp**.

87527 **1d** Deletes the first line, saving it in buffer *p*.

87528 **1d p1l** (Pee one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was  
87529 second.

### 87530 **Edit**

87531 Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,  
87532 although some (for example, **insert** and **append**) were known to confuse historical  
87533 implementations. For consistency and simplicity of specification, POSIX.1-2008 requires that any  
87534 command be supported as an argument to the **edit** command.

87535 Historically, the command argument was executed with the current line set to the last line of the  
87536 file, regardless of whether the **edit** command was executed from visual mode or not.  
87537 POSIX.1-2008 requires conformance to historical practice.

87538 Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first  
87539 `<blank>`, and there was no way to quote them. For consistency, POSIX.1-2008 requires that the  
87540 usual *ex* backslash quoting be provided.

87541 Historically, specifying the *+command* argument to the edit command required a filename to be  
 87542 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of  
 87543 specification, POSIX.1-2008 does not permit this usage to fail for that reason.

87544 Historically, only the cursor position of the last file edited was remembered by the editor.  
 87545 POSIX.1-2008 requires that this be supported; however, implementations are permitted to  
 87546 remember and restore the cursor position for any file previously edited.

#### 87547 **File**

87548 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in  
 87549 the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a  
 87550 current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-2008 does  
 87551 not permit this discrepancy, instead requiring that a message be displayed indicating that the file  
 87552 is empty.

#### 87553 **Global**

87554 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,  
 87555 only the required result of the operation.

87556 The current line and column are set as specified for the individual *ex* commands. This  
 87557 requirement is cumulative; that is, the current line and column must track across all the  
 87558 commands executed by the **global** or **v** commands.

#### 87559 **Insert**

87560 See the RATIONALE for the **append** command.

87561 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer  
 87562 was empty. POSIX.1-2008 requires that this command behave consistently with the **append**  
 87563 command.

#### 87564 **Join**

87565 The action of the **join** command in relation to the special characters is only defined for the  
 87566 POSIX locale because the correct amount of white space after a period varies; in Japanese none is  
 87567 required, in French only a single space, and so on.

#### 87568 **List**

87569 The historical output of the **list** command was potentially ambiguous. The standard developers  
 87570 believed correcting this to be more important than adhering to historical practice, and  
 87571 POSIX.1-2008 requires unambiguous output.

#### 87572 **Map**

87573 Historically, command mode maps only applied to command names; for example, if the  
 87574 character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y'  
 87575 character. POSIX.1-2008 requires this behavior. Historically, entering <control>-V as the first  
 87576 character of a *vi* command was an error. Several implementations have extended the semantics  
 87577 of *vi* such that <control>-V means that the subsequent command character is not mapped. This  
 87578 is permitted, but not required, by POSIX.1-2008. Regardless, using <control>-V to escape the  
 87579 second or later character in a sequence of characters that might match a **map** command, or any  
 87580 character in text input mode, is historical practice, and stops the entered keys from matching a  
 87581 map. POSIX.1-2008 requires conformance to historical practice.

- 87582 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored  
87583 the map. POSIX.1-2008 requires that the mapped digits not be ignored.
- 87584 The historical implementation of the **map** command did not permit **map** commands that were  
87585 more than a single character in length if the first character was printable. This behavior is  
87586 permitted, but not required, by POSIX.1-2008.
- 87587 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the  
87588 prefix of the mapped characters matched the mapping characters; for example, in the **map**:
- 87589 `:map ab abcd`
- 87590 the characters "ab" were used as is and were not remapped, but the characters "cd" were  
87591 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.  
87592 POSIX.1-2008 requires conformance to historical practice, and that such loops be interruptible.
- 87593 Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!**  
87594 command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex**  
87595 **abbreviate** command. POSIX.1-2008 requires similar modification of some historical practice for  
87596 the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.
- 87597 Historically, **maps** that were subsets of other **maps** behaved differently depending on the order  
87598 in which they were defined. For example:
- 87599 `:map! ab short`  
87600 `:map! abc long`
- 87601 would always translate the characters "ab" to "short", regardless of how fast the characters  
87602 "abc" were entered. If the entry order was reversed:
- 87603 `:map! abc long`  
87604 `:map! ab short`
- 87605 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,  
87606 and the characters might never be mapped to "short". For consistency and simplicity of  
87607 specification, POSIX.1-2008 requires that the shortest match be used at all times.
- 87608 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified  
87609 because the timing capabilities of systems are often inexact and variable, and it may depend on  
87610 other factors such as the speed of the connection. The time should be long enough for the user to  
87611 be able to complete the sequence, but not long enough for the user to have to wait. Some  
87612 implementations of *vi* have added a **keytime** option, which permits users to set the number of  
87613 0,1 seconds the editor waits for the completing characters. Because mapped terminal function  
87614 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input  
87615 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,  
87616 or, at least timed out differently.
- 87617 **Mark**
- 87618 Historically, users were able to set the "previous context" marks explicitly. In addition, the **ex**  
87619 commands `"` and `'` and the *vi* commands `"`, `'`, `,"`, and `'` all referred to the same mark. In addition,  
87620 the previous context marks were not set if the command, with which the address setting the  
87621 mark was associated, failed. POSIX.1-2008 requires conformance to historical practice.  
87622 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the  
87623 change was undone. POSIX.1-2008 requires conformance to historical practice.
- 87624 The description of the special events that set the `'` and `'` marks matches historical practice. For  
87625 example, historically the command `/a/`/`b/` did not set the `'` and `'` marks, but the command

87626 **/a/,b/delete** did.

### 87627 **Next**

87628 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,  
87629 although some (for example, **insert** and **append**) were known to confuse historical  
87630 implementations. POSIX.1-2008 requires that any command be permitted and that it behave as  
87631 specified. The **next** command can accept more than one file, so usage such as:

87632 `next `ls [abc] ``

87633 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect  
87634 only one filename.

87635 Historically, the **next** command behaved differently from the **:rewind** command in that it  
87636 ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-2008 does not  
87637 permit this behavior.

87638 Historically, the **next** command positioned the cursor as if the file had never been edited before,  
87639 regardless. POSIX.1-2008 does not permit this behavior, for consistency with the **edit** command.

87640 Implementations wanting to provide a counterpart to the **next** command that edited the  
87641 previous file have used the command **prev[ious]**, which takes no *file* argument. POSIX.1-2008  
87642 does not require this command.

### 87643 **Open**

87644 Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-2008  
87645 does not mention the **open** edit option and does not require this behavior. Some historical  
87646 implementations do not permit entering open mode from open or visual mode, only from *ex*  
87647 mode. For consistency, POSIX.1-2008 does not permit this behavior.

87648 Historically, entering open mode from the command line (that is, *vi +open*) resulted in  
87649 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command  
87650 <control>-G did not work. For consistency, POSIX.1-2008 does not permit this behavior.

87651 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern  
87652 delimiter. For consistency, POSIX.1-2008 requires that the search delimiters used by the **s**, **global**,  
87653 and **v** commands be accepted as well.

### 87654 **Preserve**

87655 The **preserve** command does not historically cause the file to be considered unmodified for the  
87656 purposes of future commands that may exit the editor. POSIX.1-2008 requires conformance to  
87657 historical practice.

87658 Historical documentation stated that mail was not sent to the user when preserve was executed;  
87659 however, historical implementations did send mail in this case. POSIX.1-2008 requires  
87660 conformance to the historical implementations.

87661 **Print**

87662 The writing of NUL by the **print** command is not specified as a special case because the standard  
87663 developers did not want to require *ex* to support NUL characters. Historically, characters were  
87664 displayed using the ARPA standard mappings, which are as follows:

- 87665 1. Printable characters are left alone.
- 87666 2. Control characters less than \177 are represented as '^' followed by the character offset  
87667 from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
- 87668 3. \177 is represented as '^?' followed by '?'. .

87669 The display of characters having their eighth bit set was less standard. Existing implementations  
87670 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their  
87671 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)  
87672 The latter probably has the best claim to historical practice because it was used for the **-v** option  
87673 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

87674 No specific display format is required by POSIX.1-2008.

87675 Explicit dependence on the ASCII character set has been avoided where possible, hence the use  
87676 of the phrase an "implementation-defined multi-character sequence" for the display of non-  
87677 printable characters in preference to the historical usage of, for instance, "^I" for the <tab>.  
87678 Implementations are encouraged to conform to historical practice in the absence of any strong  
87679 reason to diverge.

87680 Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized  
87681 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command  
87682 names. POSIX.1-2008 permits, but does not require, this historical practice because capital forms  
87683 of the commands are used by some implementations for other purposes.

87684 **Put**

87685 Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open  
87686 or visual mode **P** command, if the buffer was named and was cut in character mode, and the  
87687 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer  
87688 was the source of the text, the entire line from which the text was taken was usually **put**, and the  
87689 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.  
87690 In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in  
87691 errors as well, such as appending text that was unrelated to the (supposed) contents of the  
87692 buffer. For consistency and simplicity of specification, POSIX.1-2008 does not permit these  
87693 behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the  
87694 buffers are not altered by changing the mode of the editor.

87695 **Read**

87696 Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file,  
87697 left an empty line as the first line of the file. For consistency and simplicity of specification,  
87698 POSIX.1-2008 does not permit this behavior. Historically, a **read** in open or visual mode from a  
87699 program left the cursor at the last line read in, not the first. For consistency, POSIX.1-2008 does  
87700 not permit this behavior.

87701 Historical implementations of *ex* were unable to undo **read** commands that read from the output  
87702 of a program. For consistency, POSIX.1-2008 does not permit this behavior.

87703 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified  
87704 "characters", not "bytes". POSIX.1-2008 requires that the number of bytes be displayed, not the

87705 number of characters, because it may be difficult in multi-byte implementations to determine the  
87706 number of characters read. Implementations are encouraged to clarify the message displayed to  
87707 the user.

87708 Historically, reads were not permitted on files other than type regular, except that FIFO files  
87709 could be read (probably only because they did not exist when *ex* and *vi* were originally written).  
87710 Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way  
87711 to force the read. POSIX.1-2008 permits, but does not require, this behavior.

#### 87712 **Recover**

87713 Some historical implementations of the editor permitted users to recover the edit buffer contents  
87714 from a previous edit session, and then exit without saving those contents (or explicitly  
87715 discarding them). The intent of POSIX.1-2008 in requiring that the edit buffer be treated as  
87716 already modified is to prevent this user error.

#### 87717 **Rewind**

87718 Historical implementations supported the **rewind** command when the user was editing the first  
87719 file in the list; that is, the file that the **rewind** command would edit. POSIX.1-2008 requires  
87720 conformance to historical practice.

#### 87721 **Substitute**

87722 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the  
87723 last regular expression used in any command as the pattern, the same as the **~** command. The **r**  
87724 option is not required by POSIX.1-2008. Historically, the **c** and **g** options were toggled; for  
87725 example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of  
87726 specification, POSIX.1-2008 does not permit this behavior.

87727 The tilde command is often used to replace the last search RE. For example, in the sequence:

```
87728 s/red/blue/
87729 /green
87730 ~
```

87731 the **~** command is equivalent to:

```
87732 s/green/blue/
```

87733 Historically, *ex* accepted all of the following forms:

```
87734 s/abc/def/
87735 s/abc/def
87736 s/abc/
87737 s/abc
```

87738 POSIX.1-2008 requires conformance to this historical practice.

87739 The **s** command presumes that the **'^'** character only occupies a single column in the display.  
87740 Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in  
87741 the display. There are no known character sets for which this is not true.

87742 Historically, the final column position for the substitute commands was based on previous  
87743 column movements; a search for a pattern followed by a substitution would leave the column  
87744 position unchanged, while a **0** command followed by a substitution would change the column  
87745 position to the first non-**<blank>**. For consistency and simplicity of specification, POSIX.1-2008  
87746 requires that the final column position always be set to the first non-**<blank>**.

- 87747           **Set**
- 87748           Historical implementations redisplayed all of the options for each occurrence of the **all** keyword.  
87749           POSIX.1-2008 permits, but does not require, this behavior.
- 87750           **Tag**
- 87751           No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry.  
87752           Historical practice has been to look for the path found in the **tags** file, based on the current  
87753           directory. A useful extension found in some implementations is to look based on the directory  
87754           containing the tags file that held the entry, as well. No requirement is made as to which reference  
87755           for the tag in the tags file is used. This is deliberate, in order to permit extensions such as  
87756           multiple entries in a tags file for a tag.
- 87757           Because users often specify many different tags files, some of which need not be relevant or exist  
87758           at any particular time, POSIX.1-2008 requires that error messages about problem tags files be  
87759           displayed only if the requested tag is not found, and then, only once for each time that the **tag**  
87760           edit option is changed.
- 87761           The requirement that the current edit buffer be unmodified is only necessary if the file indicated  
87762           by the tag entry is not the same as the current file (as defined by the current pathname).  
87763           Historically, the file would be reloaded if the filename had changed, as well as if the filename  
87764           was different from the current pathname. For consistency and simplicity of specification,  
87765           POSIX.1-2008 does not permit this behavior, requiring that the name be the only factor in the  
87766           decision.
- 87767           Historically, *vi* only searched for tags in the current file from the current cursor to the end of the  
87768           file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor  
87769           were not found. POSIX.1-2008 considers this a bug, and implementations are required to search  
87770           for the first occurrence in the file, regardless.
- 87771           **Undo**
- 87772           The **undo** description deliberately uses the word “modified”. The **undo** command is not  
87773           intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**,  
87774           or **recover**.
- 87775           Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes  
87776           attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and  
87777           sometimes, in the presence of maps, placing the cursor on the last line added or changed instead  
87778           of the first. POSIX.1-2008 requires a simplified behavior for consistency and simplicity of  
87779           specification.
- 87780           **Version**
- 87781           The **version** command cannot be exactly specified since there is no widely-accepted definition of  
87782           what the version information should contain. Implementations are encouraged to do something  
87783           reasonably intelligent.

87784 **Write**

87785 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified  
 87786 “characters”, not “bytes”. POSIX.1-2008 requires that the number of bytes be displayed, not the  
 87787 number of characters because it may be difficult in multi-byte implementations to determine the  
 87788 number of characters written. Implementations are encouraged to clarify the message displayed  
 87789 to the user.

87790 Implementation-defined tests are permitted so that implementations can make additional  
 87791 checks; for example, for locks or file modification times.

87792 Historically, attempting to append to a nonexistent file caused an error. It has been left  
 87793 unspecified in POSIX.1-2008 to permit implementations to let the **write** succeed, so that the  
 87794 append semantics are similar to those of the historical *cs/h*.

87795 Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around  
 87796 dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote  
 87797 them as files of a single, empty line. POSIX.1-2008 does not permit this behavior.

87798 Historically, *ex* restored standard output and standard error to their values as of when *ex* was  
 87799 invoked, before writes to programs were performed. This could disturb the terminal  
 87800 configuration as well as be a security issue for some terminals. POSIX.1-2008 does not permit  
 87801 this, requiring that the program output be captured and displayed as if by the *ex print*  
 87802 command.

87803 **Adjust Window**

87804 Historically, the line count was set to the value of the **scroll** option if the type character was end-  
 87805 of-file. This feature was broken on most historical implementations long ago, however, and is  
 87806 not documented anywhere. For this reason, POSIX.1-2008 is resolutely silent.

87807 Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+**  
 87808 and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =**  
 87809 were historically invalid.) POSIX.1-2008 requires conformance to this historical practice.

87810 Historically, the **z** command was further <blank>-sensitive in that the *count* could not be  
 87811 <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the  
 87812 *count* is not ambiguous with respect to either the type character or the flags, this is not permitted  
 87813 by POSIX.1-2008.

87814 **Escape**

87815 Historically, *ex* filter commands only read the standard output of the commands, letting  
 87816 standard error appear on the terminal as usual. The *vi* utility, however, read both standard  
 87817 output and standard error. POSIX.1-2008 requires the latter behavior for both *ex* and *vi*, for  
 87818 consistency.

87819 **Shift Left and Shift Right**

87820 Historically, it was possible to add shift characters to increase the effect of the command; for  
 87821 example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default  
 87822 1. POSIX.1-2008 requires conformance to historical practice.

- 87823            **<control>-D**
- 87824            Historically, the **<control>-D** command erased the prompt, providing the user with an unbroken  
87825            presentation of lines from the edit buffer. This is not required by POSIX.1-2008; implementations  
87826            are encouraged to provide it if possible. Historically, the **<control>-D** command took, and then  
87827            ignored, a *count*. POSIX.1-2008 does not permit this behavior.
- 87828            **Write Line Number**
- 87829            Historically, the **ex =** command, when executed in *ex* mode in an empty edit buffer, reported 0,  
87830            and from open or visual mode, reported 1. For consistency and simplicity of specification,  
87831            POSIX.1-2008 does not permit this behavior.
- 87832            **Execute**
- 87833            Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,  
87834            **insert**, and **change**) in executed buffers. POSIX.1-2008 does not permit this exclusion for  
87835            consistency.
- 87836            Historically, the logical contents of the buffer being executed did not change if the buffer itself  
87837            were modified by the commands being executed; that is, buffer execution did not support self-  
87838            modifying code. POSIX.1-2008 requires conformance to historical practice.
- 87839            Historically, the **@** command took a range of lines, and the **@** buffer was executed once per line,  
87840            with the current line ( *' . '* ) set to each specified line. POSIX.1-2008 requires conformance to  
87841            historical practice.
- 87842            Some historical implementations did not notice if errors occurred during buffer execution. This,  
87843            coupled with the ability to specify a range of lines for the *ex @* command, makes it trivial to  
87844            cause them to drop **core**. POSIX.1-2008 requires that implementations stop buffer execution if  
87845            any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are  
87846            replaced (for example, the buffer executes the *ex :edit* command).
- 87847            **Regular Expressions in ex**
- 87848            Historical practice is that the characters in the replacement part of the last **s** command—that is,  
87849            those matched by entering a *' ~ '* in the regular expression—were not further expanded by the  
87850            regular expression engine. So, if the characters contained the string "a . , " they would match  
87851            *' a '* followed by ". , " and not *' a '* followed by any character. POSIX.1-2008 requires  
87852            conformance to historical practice.
- 87853            **Edit Options in ex**
- 87854            The following paragraphs describe the historical behavior of some edit options that were not, for  
87855            whatever reason, included in POSIX.1-2008. Implementations are strongly encouraged to only  
87856            use these names if the functionality described here is fully supported.
- 87857            **extended**    The **extended** edit option has been used in some implementations of *vi* to provide  
87858            extended regular expressions instead of basic regular expressions This option was  
87859            omitted from POSIX.1-2008 because it is not widespread historical practice.
- 87860            **flash**        The **flash** edit option historically caused the screen to flash instead of beeping on  
87861            error. This option was omitted from POSIX.1-2008 because it is not found in some  
87862            historical implementations.

|       |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 87863 | <b>hardtabs</b>  | The <b>hardtabs</b> edit option historically defined the number of columns between hardware tab settings. This option was omitted from POSIX.1-2008 because it was believed to no longer be generally useful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 87864 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87865 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87866 | <b>modeline</b>  | The <b>modeline</b> (sometimes named <b>modelines</b> ) edit option historically caused <i>ex</i> or <i>vi</i> to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 87867 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87868 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87869 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87870 | <b>open</b>      | The <b>open</b> edit option historically disallowed the <i>ex</i> <b>open</b> and <b>visual</b> commands. This edit option was omitted because these commands are required by POSIX.1-2008.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87871 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87872 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87873 | <b>optimize</b>  | The <b>optimize</b> edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return> characters when printing more than one logical line of output. This option was omitted from POSIX.1-2008 because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.                                                                                                                                                                                                                                                                                                                                                                      |
| 87874 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87875 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87876 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87877 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87878 | <b>ruler</b>     | The <b>ruler</b> edit option has been used in some implementations of <i>vi</i> to present a current row/column ruler for the user. This option was omitted from POSIX.1-2008 because it is not widespread historical practice.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87879 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87880 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87881 | <b>sourceany</b> | The <b>sourceany</b> edit option historically caused <i>ex</i> or <i>vi</i> to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 87882 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87883 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87884 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87885 | <b>timeout</b>   | The <b>timeout</b> edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-2008 because its behavior is now standard, it is not widely useful, and it was rarely documented.                                                                                                                                                                                                                                                                                                                                                                                                               |
| 87886 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87887 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87888 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87889 | <b>verbose</b>   | The <b>verbose</b> edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option <b>terse</b> did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-2008 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users. |
| 87890 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87891 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87892 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87893 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87894 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87895 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87896 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87897 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87898 | <b>wraplen</b>   | The <b>wraplen</b> edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from POSIX.1-2008 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.                                                                                                                                                                                                                                                                                  |
| 87899 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87900 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87901 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87902 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 87903 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

87904 **autoindent, ai**

87905 Historically, the command **0a** did not do any autoindentation, regardless of the current  
87906 indentation of line 1. POSIX.1-2008 requires that any indentation present in line 1 be used.

87907 **autoprint, ap**

87908 Historically, the **autoprint** edit option was not completely consistent or based solely on  
87909 modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,  
87910 but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not  
87911 affected by **autoprint**, and the **tag** command, which was affected by **autoprint**. POSIX.1-2008  
87912 requires conformance to historical practice.

87913 Historically, the **autoprint** option only applied to the last of multiple commands entered using  
87914 <vertical-line> delimiters; for example, **delete** <newline> was affected by **autoprint**, but  
87915 **delete | version** <newline> was not. POSIX.1-2008 requires conformance to historical practice.

87916 **autowrite, aw**

87917 Appending the '!' character to the *ex* **next** command to avoid performing an automatic write  
87918 was not supported in historical implementations. POSIX.1-2008 requires that the behavior match  
87919 the other *ex* commands for consistency.

87920 **ignorecase, ic**

87921 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to  
87922 counter-intuitive situations when uppercase characters were used in range expressions.  
87923 Historically, the process was as follows:

- 87924 1. Take a line of text from the edit buffer.
- 87925 2. Convert uppercase to lowercase in text line.
- 87926 3. Convert uppercase to lowercase in regular expressions, except in character class  
87927 specifications.
- 87928 4. Match regular expressions against text.

87929 This would mean that, with **ignorecase** in effect, the text:

87930 The cat sat on the mat

87931 would be matched by

87932 /<sup>^</sup>the/

87933 but not by:

87934 /<sup>^</sup>[A-Z]he/

87935 For consistency with other commands implementing regular expressions, POSIX.1-2008 does not  
87936 permit this behavior.

87937 **paragraphs, para**

87938 The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options  
 87939 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text  
 87940 formatter, and a “portable user” could use the {, }, [[, ]], (, and ) commands in open or visual  
 87941 mode and have the cursor stop in unexpected places. POSIX.1-2008 specifies their values in the  
 87942 POSIX locale because the unusual grouping (they only work when grouped into two characters  
 87943 at a time) means that they cannot be used for general-purpose movement, regardless.

87944 **readonly**

87945 Implementations are encouraged to provide the best possible information to the user as to the  
 87946 read-only status of the file, with the exception that they should not consider the current special  
 87947 privileges of the process. This provides users with a safety net because they must force the  
 87948 overwrite of read-only files, even when running with additional privileges.

87949 The **readonly** edit option specification largely conforms to historical practice. The only  
 87950 difference is that historical implementations did not notice that the user had set the **readonly**  
 87951 edit option in cases where the file was already marked read-only for some reason, and would  
 87952 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were  
 87953 replaced. This behavior is disallowed by POSIX.1-2008.

87954 **report**

87955 The requirement that lines copied to a buffer interact differently than deleted lines is historical  
 87956 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be  
 87957 written, but 4 lines must be copied before a report is written.

87958 The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based  
 87959 on the total number of lines added or deleted during the command execution, and that  
 87960 commands executed by the **global** and **v** commands not present reports, is historical practice.  
 87961 POSIX.1-2008 extends historical practice by requiring that buffer execution be treated similarly.  
 87962 The reasons for this are two-fold. Historically, only the report by the last command executed  
 87963 from the buffer would be seen by the user, as each new report would overwrite the last. In  
 87964 addition, the standard developers believed that buffer execution had more in common with  
 87965 **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for  
 87966 consistency and simplicity of specification.

87967 **showmatch, sm**

87968 The length of time the cursor spends on the matching character is unspecified because the  
 87969 timing capabilities of systems are often inexact and variable. The time should be long enough for  
 87970 the user to notice, but not long enough for the user to become annoyed. Some implementations  
 87971 of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals  
 87972 the cursor pauses on the matching character.

87973 **showmode**

87974 The **showmode** option has been used in some historical implementations of *ex* and *vi* to display  
 87975 the current editing mode when in open or visual mode. The editing modes have generally  
 87976 included “command” and “input”, and sometimes other modes such as “replace” and  
 87977 “change”. The string was usually displayed on the bottom line of the screen at the far right-hand  
 87978 corner. In addition, a preceding ‘\*’ character often denoted whether the contents of the edit  
 87979 buffer had been modified. The latter display has sometimes been part of the **showmode** option,  
 87980 and sometimes based on another option. This option was not available in the 4 BSD historical

87981 implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is  
87982 required by POSIX.1-2008.

87983 The **smd** shorthand for the **showmode** option was not present in all historical implementations  
87984 of the editor. POSIX.1-2008 requires it, for consistency.

87985 Not all historical implementations of the editor displayed a mode string for command mode,  
87986 differentiating command mode from text input mode by the absence of a mode string.  
87987 POSIX.1-2008 permits this behavior for consistency with historical practice, but implementations  
87988 are encouraged to provide a display string for both modes.

#### 87989 **slowopen**

87990 Historically, the **slowopen** option was automatically set if the terminal baud rate was less than  
87991 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**  
87992 option had two effects. First, when inserting characters in the middle of a line, characters after  
87993 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when  
87994 creating a new line of text, lines after the current line would not be scrolled down, but would  
87995 appear to be overwritten. In both cases, ending text input mode would cause the screen to be  
87996 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently  
87997 intelligent caused the editor to ignore the **slowopen** option. POSIX.1-2008 permits most  
87998 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option  
87999 is set by the user.

#### 88000 **tags**

88001 The default path for tags files is left unspecified as implementations may have their own **tags**  
88002 implementations that do not correspond to the historical ones. The default **tags** option value  
88003 should probably at least include the file **./tags**.

#### 88004 **term**

88005 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial  
88006 terminal information was loaded. This is permitted by POSIX.1-2008; however, implementations  
88007 are encouraged to permit the user to modify their terminal type at any time.

#### 88008 **terse**

88009 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,  
88010 for some error messages. This is permitted, but not required, by POSIX.1-2008. Historically, most  
88011 common visual mode errors (for example, trying to move the cursor past the end of a line) did  
88012 not result in an error message, but simply alerted the terminal. Implementations wishing to  
88013 provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not  
88014 **terse**.

#### 88015 **window**

88016 In historical implementations, the default for the **window** edit option was based on the baud  
88017 rate as follows:

88018 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for  
88019 example, the line:

88020 `set w300=12`

88021 would set the window option to 12 if the baud rate was less than 1 200.

88022 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

88023 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

88024 The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-2008 because of their  
88025 dependence on specific baud rates.

88026 In historical implementations, the size of the window displayed by various commands was  
88027 related to, but not necessarily the same as, the **window** edit option. For example, the size of the  
88028 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**  
88029 edit option. However, changing the value of the **window** edit option did change the number of  
88030 lines that were displayed when the screen was repainted. POSIX.1-2008 does not permit this  
88031 behavior in the interests of consistency and simplicity of specification, and requires that all  
88032 commands that change the number of lines that are displayed do it by setting the value of the  
88033 **window** edit option.

#### 88034 **wrapmargin, wm**

88035 Historically, the **wrapmargin** option did not affect maps inserting characters that also had  
88036 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used  
88037 maps that depend on this behavior. For consistency and simplicity of specification,  
88038 POSIX.1-2008 does not permit this behavior.

88039 Historically, **wrapmargin** was calculated using the column display width of all characters on the  
88040 screen. For example, an implementation using "**^I**" to represent <tab> characters when the **list**  
88041 edit option was set, where '**^**' and '**I**' each took up a single column on the screen, would  
88042 calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option  
88043 similarly changed the effective length of the line as well. POSIX.1-2008 requires conformance to  
88044 historical practice.

88045 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
88046 but this has been modified in this version.

#### 88047 **FUTURE DIRECTIONS**

88048 None.

#### 88049 **SEE ALSO**

88050 [Section 2.9.1.1](#) (on page 2317), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

88051 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on  
88052 page 215)

88053 XSH [access\(\)](#)

#### 88054 **CHANGE HISTORY**

88055 First released in Issue 2.

#### 88056 **Issue 5**

88057 The FUTURE DIRECTIONS section is added.

#### 88058 **Issue 6**

88059 This utility is marked as part of the User Portability Utilities option.

88060 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

- 88061 The following new requirements on POSIX implementations derive from alignment with the  
88062 Single UNIX Specification:
- 88063 • In the **map** command description, the sequence *#digit* is added.
  - 88064 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.
- 88065 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This  
88066 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,  
88067 #55, #56, #57, #61, #62, #63, #64, #65, and #78.
- 88068 The **-l** option is removed.
- 88069 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.
- 88070 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial  
88071 correction in the EXTENDED DESCRIPTION.
- 88072 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing  
88073 behavior on systems with bytes consisting of more than eight bits.
- 88074 **Issue 7**
- 88075 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is  
88076 ' - ' .
- 88077 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.
- 88078 Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex write* command.
- 88079 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 88080 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88081 **NAME**88082 `expand` — convert tabs to spaces88083 **SYNOPSIS**88084 `expand [-t tablist] [file...]`88085 **DESCRIPTION**

88086 The *expand* utility shall write files or the standard input to the standard output with `<tab>`  
 88087 characters replaced with one or more `<space>` characters needed to pad to the next tab stop. Any  
 88088 `<backspace>` characters shall be copied to the output and cause the column position count for  
 88089 tab stop calculations to be decremented; the column position count shall not be decremented  
 88090 below zero.

88091 **OPTIONS**88092 The *expand* utility shall conform to XBD Section 12.2 (on page 215).

88093 The following option shall be supported:

88094 `-t tablist` Specify the tab stops. The application shall ensure that the argument *tablist* consists  
 88095 of either a single positive decimal integer or a list of tabstops. If a single number is  
 88096 given, tabs shall be set that number of column positions apart instead of the  
 88097 default 8.

88098 If a list of tabstops is given, the application shall ensure that it consists of a list of  
 88099 two or more positive decimal integers, separated by `<blank>` or `<comma>`  
 88100 characters, in ascending order. The `<tab>` characters shall be set at those specific  
 88101 column positions. Each tab stop *N* shall be an integer value greater than zero, and  
 88102 the list is in strictly ascending order. This is taken to mean that, from the start of a  
 88103 line of output, tabbing to position *N* shall cause the next character output to be in  
 88104 the (*N*+1)th column position on that line.

88105 In the event of *expand* having to process a `<tab>` at a position beyond the last of  
 88106 those specified in a multiple tab-stop list, the `<tab>` shall be replaced by a single  
 88107 `<space>` in the output.

88108 **OPERANDS**

88109 The following operand shall be supported:

88110 *file* The pathname of a text file to be used as input.88111 **STDIN**

88112 See the INPUT FILES section.

88113 **INPUT FILES**

88114 Input files shall be text files.

88115 **ENVIRONMENT VARIABLES**88116 The following environment variables shall affect the execution of *expand*:

88117 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 88118 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 88119 variables used to determine the values of locale categories.)

88120 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 88121 internationalization variables.

88122 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 88123 characters (for example, single-byte as opposed to multi-byte characters in  
 88124 arguments and input files), the processing of `<tab>` and `<space>` characters, and  
 88125 for the determination of the width in column positions each character would

**expand**

Utilities

- 88126 occupy on an output device.
- 88127 *LC\_MESSAGES*
- 88128 Determine the locale that should be used to affect the format and contents of  
88129 diagnostic messages written to standard error.
- 88130 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 88131 **ASYNCHRONOUS EVENTS**
- 88132 Default.
- 88133 **STDOUT**
- 88134 The standard output shall be equivalent to the input files with <tab> characters converted into  
88135 the appropriate number of <space> characters.
- 88136 **STDERR**
- 88137 The standard error shall be used only for diagnostic messages.
- 88138 **OUTPUT FILES**
- 88139 None.
- 88140 **EXTENDED DESCRIPTION**
- 88141 None.
- 88142 **EXIT STATUS**
- 88143 The following exit values shall be returned:
- 88144 0 Successful completion
- 88145 >0 An error occurred.
- 88146 **CONSEQUENCES OF ERRORS**
- 88147 The *expand* utility shall terminate with an error message and non-zero exit status upon  
88148 encountering difficulties accessing one of the *file* operands.
- 88149 **APPLICATION USAGE**
- 88150 None.
- 88151 **EXAMPLES**
- 88152 None.
- 88153 **RATIONALE**
- 88154 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific  
88155 columns, and so on) that contain <tab> characters.
- 88156 See XBD Section 3.103 (on page 50).
- 88157 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8  
88158 mandates that *expand* shall accept the integers (within the single argument) separated using  
88159 either <comma> or <blank> characters.
- 88160 Earlier versions of this standard allowed the following form in the SYNOPSIS:
- 88161 `expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]`
- 88162 This form is no longer specified by POSIX.1-2008 but may be present in some implementations.
- 88163 **FUTURE DIRECTIONS**
- 88164 None.

88165 **SEE ALSO**88166 *tabs, unexpand*88167 XBD [Section 3.103](#) (on page 50), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)88168 **CHANGE HISTORY**

88169 First released in Issue 4.

88170 **Issue 6**

88171 This utility is marked as part of the User Portability Utilities option.

88172 The APPLICATION USAGE section is added.

88173 The obsolescent SYNOPSIS is removed.

88174 The *LC\_CTYPE* environment variable description is updated to align with the IEEE P1003.2b  
88175 draft standard.

88176 The normative text is reworded to avoid use of the term “must” for application requirements.

88177 **Issue 7**

88178 Austin Group Interpretation 1003.1-2001 #027 is applied.

88179 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88180 The *expand* utility is moved from the User Portability Utilities option to the Base. User  
88181 Portability Utilities is now an option for interactive utilities.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**expr**

Utilities

88182 **NAME**88183 `expr` — evaluate arguments as an expression88184 **SYNOPSIS**88185 `expr operand...`88186 **DESCRIPTION**88187 The `expr` utility shall evaluate an expression and write the result to standard output.88188 **OPTIONS**

88189 None.

88190 **OPERANDS**88191 The single expression evaluated by `expr` shall be formed from the `operand` operands, as described  
88192 in the EXTENDED DESCRIPTION section. The application shall ensure that each of the  
88193 expression operator symbols:88194 `( ) | & = > >= < <= != + - * / % :`88195 and the symbols *integer* and *string* in the table are provided as separate arguments to `expr`.88196 **STDIN**

88197 Not used.

88198 **INPUT FILES**

88199 None.

88200 **ENVIRONMENT VARIABLES**88201 The following environment variables shall affect the execution of `expr`:88202 `LANG` Provide a default value for the internationalization variables that are unset or null.  
88203 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
88204 variables used to determine the values of locale categories.)88205 `LC_ALL` If set to a non-empty string value, override the values of all the other  
88206 internationalization variables.88207 `LC_COLLATE`88208 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
88209 character collating elements within regular expressions and by the string  
88210 comparison operators.88211 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
88212 characters (for example, single-byte as opposed to multi-byte characters in  
88213 arguments) and the behavior of character classes within regular expressions.88214 `LC_MESSAGES`88215 Determine the locale that should be used to affect the format and contents of  
88216 diagnostic messages written to standard error.88217 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.88218 **ASYNCHRONOUS EVENTS**

88219 Default.

88220 **STDOUT**88221 The `expr` utility shall evaluate the expression and write the result, followed by a <newline>, to  
88222 standard output.

88223 **STDERR**

88224 The standard error shall be used only for diagnostic messages.

88225 **OUTPUT FILES**

88226 None.

88227 **EXTENDED DESCRIPTION**

88228 The formation of the expression to be evaluated is shown in the following table. The symbols  
 88229 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the  
 88230 expression operator symbols (all separate arguments) by recursive application of the constructs  
 88231 described in the table. The expressions are listed in order of increasing precedence, with equal-  
 88232 precedence operators grouped between horizontal lines. All of the operators shall be left-  
 88233 associative.

| Expression                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expr1</i>   <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.                                                                                                                                                                                                                                               |
| <i>expr1</i> & <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.                                                                                                                                                                                                                                                                                           |
| <i>expr1</i> = <i>expr2</i><br><i>expr1</i> > <i>expr2</i><br><i>expr1</i> >= <i>expr2</i><br><i>expr1</i> < <i>expr2</i><br><i>expr1</i> <= <i>expr2</i><br><i>expr1</i> != <i>expr2</i> | Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false.<br>Equal.<br>Greater than.<br>Greater than or equal.<br>Less than.<br>Less than or equal.<br>Not equal. |
| <i>expr1</i> + <i>expr2</i><br><i>expr1</i> - <i>expr2</i>                                                                                                                                | Addition of decimal integer-valued arguments.<br>Subtraction of decimal integer-valued arguments.                                                                                                                                                                                                                                                                                                          |
| <i>expr1</i> * <i>expr2</i><br><i>expr1</i> / <i>expr2</i><br><i>expr1</i> % <i>expr2</i>                                                                                                 | Multiplication of decimal integer-valued arguments.<br>Integer division of decimal integer-valued arguments, producing an integer result.<br>Remainder of integer division of decimal integer-valued arguments.                                                                                                                                                                                            |
| <i>expr1</i> : <i>expr2</i>                                                                                                                                                               | Matching expression; see below.                                                                                                                                                                                                                                                                                                                                                                            |
| ( <i>expr</i> )                                                                                                                                                                           | Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.                                                                                                                                                                                                                                                                                |
| <i>integer</i>                                                                                                                                                                            | An argument consisting only of an (optional) unary minus followed by digits.                                                                                                                                                                                                                                                                                                                               |
| <i>string</i>                                                                                                                                                                             | A string argument; see below.                                                                                                                                                                                                                                                                                                                                                                              |

88265 **Matching Expression**

88266 The `' : '` matching operator shall compare the string resulting from the evaluation of *expr1* with  
 88267 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax  
 88268 shall be that defined in XBD Section 9.3 (on page 183), except that all patterns are anchored to  
 88269 the beginning of the string (that is, only sequences starting at the first character of a string are  
 88270 matched by the regular expression) and, therefore, it is unspecified whether `' ^ '` is a special  
 88271 character in that context. Usually, the matching operator shall return a string representing the  
 88272 number of characters matched (`' 0 '` on failure). Alternatively, if the pattern contains at least one  
 88273 regular expression subexpression `"[\(.\.\.)]"`, the string matched by the back-reference  
 88274 expression `"\1"` shall be returned. If the back-reference expression `"\1"` does not match, then  
 88275 the null string shall be returned.

88276 **String Operand**

88277 A string argument is an argument that cannot be identified as an *integer* argument or as one of  
 88278 the expression operator symbols shown in the OPERANDS section.

88279 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

88280 **EXIT STATUS**

88281 The following exit values shall be returned:

- 88282 0 The *expression* evaluates to neither null nor zero.
- 88283 1 The *expression* evaluates to null or zero.
- 88284 2 Invalid *expression*.
- 88285 >2 An error occurred.

88286 **CONSEQUENCES OF ERRORS**

88287 Default.

88288 **APPLICATION USAGE**

88289 After argument processing by the shell, *expr* is not required to be able to tell the difference  
 88290 between an operator and an operand except by the value. If `"$a"` is `' = '`, the command:

88291 `expr $a = '='`

88292 looks like:

88293 `expr = = =`

88294 as the arguments are passed to *expr* (and they all may be taken as the `' = '` operator). The  
 88295 following works reliably:

88296 `expr X$a = X=`

88297 Also note that this volume of POSIX.1-2008 permits implementations to extend utilities. The *expr*  
 88298 utility permits the integer arguments to be preceded with a unary minus. This means that an  
 88299 integer argument could look like an option. Therefore, the conforming application must employ  
 88300 the `"--"` construct of Guideline 10 of XBD Section 12.2 (on page 215) to protect its operands if  
 88301 there is any chance the first operand might be a negative integer (or any string with a leading  
 88302 minus).

88303 **EXAMPLES**88304 The *expr* utility has a rather difficult syntax:

- 88305 • Many of the operators are also shell control operators or reserved words, so they have to  
88306 be escaped on the command line.
- 88307 • Each part of the expression is composed of separate arguments, so liberal usage of <blank>  
88308 characters is required. For example:

| Invalid                 | Valid                        |
|-------------------------|------------------------------|
| <i>expr</i> 1+2         | <i>expr</i> 1 + 2            |
| <i>expr</i> "1 + 2"     | <i>expr</i> 1 + 2            |
| <i>expr</i> 1 + (2 * 3) | <i>expr</i> 1 + \( 2 \* 3 \) |

88313 In many cases, the arithmetic and string features provided as part of the shell command  
88314 language are easier to use than their equivalents in *expr*. Newly written scripts should avoid  
88315 *expr* in favor of the new features within the shell; see [Section 2.5](#) (on page 2301) and [Section 2.6.4](#)  
88316 (on page 2310).

88317 The following command:

88318 `a=$(expr $a + 1)`88319 adds 1 to the variable *a*.88320 The following command, for "*\$a*" equal to either `/usr/abc/file` or just `file`:88321 `expr $a : '.*\/(.*\)' \| $a`

88322 returns the last segment of a pathname (that is, `file`). Applications should avoid the character  
88323 `/` used alone as an argument; *expr* may interpret it as the division operator.

88324 The following command:

88325 `expr "//$a" : '.*\/(.*\)'`

88326 is a better representation of the previous example. The addition of the `"/"` characters  
88327 eliminates any ambiguity about the division operator and simplifies the whole expression. Also  
88328 note that pathnames may contain characters contained in the *IFS* variable and should be quoted  
88329 to avoid having "*\$a*" expand into multiple arguments.

88330 The following command:

88331 `expr "$VAR" : '.*'`88332 returns the number of characters in *VAR*.88333 **RATIONALE**

88334 In an early proposal, EREs were used in the matching expression syntax. This was changed to  
88335 BREs to avoid breaking historical applications.

88336 The use of a leading <circumflex> in the BRE is unspecified because many historical  
88337 implementations have treated it as a special character, despite their system documentation. For  
88338 example:

88339 `expr foo : ^foo`      `expr ^foo : ^foo`

88340 return 3 and 0, respectively, on those systems; their documentation would imply the reverse.  
88341 Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on  
88342 this undocumented feature.

88343 **FUTURE DIRECTIONS**

88344 None.

88345 **SEE ALSO**88346 [Section 2.5](#) (on page 2301), [Section 2.6.4](#) (on page 2310)88347 XBD [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on page 215)88348 **CHANGE HISTORY**

88349 First released in Issue 2.

88350 **Issue 5**

88351 The FUTURE DIRECTIONS section is added.

88352 **Issue 6**88353 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE  
88354 PASC Interpretation 1003.2 #104.

88355 The normative text is reworded to avoid use of the term “must” for application requirements.

88356 **Issue 7**

88357 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

88358 The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of  
88359 the operands is *operand*.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*Utilities***false**

- 88360 **NAME**  
88361 false — return false value
- 88362 **SYNOPSIS**  
88363 false
- 88364 **DESCRIPTION**  
88365 The *false* utility shall return with a non-zero exit code.
- 88366 **OPTIONS**  
88367 None.
- 88368 **OPERANDS**  
88369 None.
- 88370 **STDIN**  
88371 Not used.
- 88372 **INPUT FILES**  
88373 None.
- 88374 **ENVIRONMENT VARIABLES**  
88375 None.
- 88376 **ASYNCHRONOUS EVENTS**  
88377 Default.
- 88378 **STDOUT**  
88379 Not used.
- 88380 **STDERR**  
88381 Not used.
- 88382 **OUTPUT FILES**  
88383 None.
- 88384 **EXTENDED DESCRIPTION**  
88385 None.
- 88386 **EXIT STATUS**  
88387 The *false* utility shall always exit with a value other than zero.
- 88388 **CONSEQUENCES OF ERRORS**  
88389 Default.
- 88390 **APPLICATION USAGE**  
88391 None.
- 88392 **EXAMPLES**  
88393 None.
- 88394 **RATIONALE**  
88395 None.
- 88396 **FUTURE DIRECTIONS**  
88397 None.
- 88398 **SEE ALSO**  
88399 *true*

**false**

*Utilities*

88400 **CHANGE HISTORY**

88401 First released in Issue 2.

88402 **Issue 6**

88403 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR  
88404 section from "None." to "Not used." for alignment with [Section 1.4](#) (on page 2288).

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

88405 **NAME**88406 `fc` — process the command history list88407 **SYNOPSIS**88408 UP `fc [-r] [-e editor] [first [last]]`88409 `fc -l [-nr] [first [last]]`88410 `fc -s [old=new] [first]`88411 **DESCRIPTION**88412 The `fc` utility shall list, or shall edit and re-execute, commands previously entered to an  
88413 interactive `sh`.

88414 The command history list shall reference commands by number. The first number in the list is  
88415 selected arbitrarily. The relationship of a number to its command shall not change except when  
88416 the user logs in and no other process is accessing the list, at which time the system may reset the  
88417 numbering to start the oldest retained command at another number (usually 1). When the  
88418 number reaches an implementation-defined upper limit, which shall be no smaller than the  
88419 value in `HISTSIZE` or 32767 (whichever is greater), the shell may wrap the numbers, starting the  
88420 next command with a lower number (usually 1). However, despite this optional wrapping of  
88421 numbers, `fc` shall maintain the time-ordering sequence of the commands. For example, if four  
88422 commands in sequence are given the numbers 32766, 32767, 1 (wrapped), and 2 as they are  
88423 executed, command 32767 is considered the command previous to 1, even though its number is  
88424 higher.

88425 When commands are edited (when the `-l` option is not specified), the resulting lines shall be  
88426 entered at the end of the history list and then re-executed by `sh`. The `fc` command that caused the  
88427 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this  
88428 shall suppress the entry into the history list and the command re-execution. Any command line  
88429 variable assignments or redirection operators used with `fc` shall affect both the `fc` command itself  
88430 as well as the command that results; for example:

88431 `fc -s -- -l 2>/dev/null`88432 reinvoke the previous command, suppressing standard error for both `fc` and the previous  
88433 command.88434 **OPTIONS**88435 The `fc` utility shall conform to XBD Section 12.2 (on page 215).

88436 The following options shall be supported:

88437 **-e** *editor* Use the editor named by *editor* to edit the commands. The *editor* string is a utility  
88438 name, subject to search via the `PATH` variable (see XBD Chapter 8, on page 173).  
88439 The value in the `FCEDIT` variable shall be used as a default when `-e` is not  
88440 specified. If `FCEDIT` is null or unset, *ed* shall be used as the editor.

88441 **-l** (The letter ell.) List the commands rather than invoking an editor on them. The  
88442 commands shall be written in the sequence indicated by the *first* and *last* operands,  
88443 as affected by `-r`, with each command preceded by the command number.

88444 **-n** Suppress command numbers when listing with `-l`.

88445 **-r** Reverse the order of the commands listed (with `-l`) or edited (with neither `-l` nor  
88446 `-s`).

88447            -s            Re-execute the command without invoking an editor.

88448   **OPERANDS**

88449            The following operands shall be supported:

88450            *first, last*       Select the commands to list or edit. The number of previous commands that can be  
88451                                   accessed shall be determined by the value of the *HISTSIZE* variable. The value of  
88452                                   *first* or *last* or both shall be one of the following:

88453            [+]*number*       A positive number representing a command number; command  
88454                                   numbers can be displayed with the -I option.

88455            -*number*       A negative decimal number representing the command that was  
88456                                   executed *number* of commands previously. For example, -1 is the  
88457                                   immediately previous command.

88458            *string*         A string indicating the most recently entered command that begins  
88459                                   with that string. If the *old=new* operand is not also specified with -s,  
88460                                   the string form of the *first* operand cannot contain an embedded  
88461                                   <equals-sign>.

88462            When the synopsis form with -s is used:

- 88463                   • If *first* is omitted, the previous command shall be used.

88464            For the synopsis forms without -s:

- 88465                   • If *last* is omitted, *last* shall default to the previous command when -I is  
88466                                   specified; otherwise, it shall default to *first*.

- 88467                   • If *first* and *last* are both omitted, the previous 16 commands shall be listed or  
88468                                   the previous single command shall be edited (based on the -I option).

- 88469                   • If *first* and *last* are both present, all of the commands from *first* to *last* shall be  
88470                                   edited (without -I) or listed (with -I). Editing multiple commands shall be  
88471                                   accomplished by presenting to the editor all of the commands at one time,  
88472                                   each command starting on a new line. If *first* represents a newer command  
88473                                   than *last*, the commands shall be listed or edited in reverse sequence,  
88474                                   equivalent to using -r. For example, the following commands on the first  
88475                                   line are equivalent to the corresponding commands on the second:

88476                   fc -r 10 20       fc     30 40

88477                   fc     20 10       fc -r 40 30

- 88478                   • When a range of commands is used, it shall not be an error to specify *first* or  
88479                                   *last* values that are not in the history list; *fc* shall substitute the value  
88480                                   representing the oldest or newest command in the list, as appropriate. For  
88481                                   example, if there are only ten commands in the history list, numbered 1 to 10:

88482                   fc -l

88483                   fc 1 99

88484                   shall list and edit, respectively, all ten commands.

88485            *old=new*       Replace the first occurrence of string *old* in the commands to be re-executed by the  
88486                                   string *new*.

88487 **STDIN**

88488 Not used.

88489 **INPUT FILES**

88490 None.

88491 **ENVIRONMENT VARIABLES**88492 The following environment variables shall affect the execution of *fc*:

88493 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for  
 88494 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
 88495 used as the editor.

88496 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 88497 not set, the shell may attempt to access or create a file *.sh\_history* in the directory  
 88498 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 88499 and write access to, or create, the history file, it shall use an unspecified  
 88500 mechanism that allows the history to operate properly. (References to history "file"  
 88501 in this section shall be understood to mean this unspecified mechanism in such  
 88502 cases.) An implementation may choose to access this variable only when  
 88503 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 88504 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 88505 by the user, the file named by the *ENV* variable, or implementation-defined system  
 88506 start-up files. In some historical shells, the history file is initialized just after the  
 88507 *ENV* file has been processed. Therefore, it is implementation-defined whether  
 88508 changes made to *HISTFILE* after the history file has been initialized are effective.  
 88509 Implementations may choose to disable the history list mechanism for users with  
 88510 appropriate privileges who do not set *HISTFILE*; the specific circumstances under  
 88511 which this occurs are implementation-defined. If more than one instance of the  
 88512 shell is using the same history file, it is unspecified how updates to the history file  
 88513 from those shells interact. As entries are deleted from the history file, they shall be  
 88514 deleted oldest first. It is unspecified when history file entries are physically  
 88515 removed from the history file.

88516 *HISTSIZE* Determine a decimal number representing the limit to the number of previous  
 88517 commands that are accessible. If this variable is unset, an unspecified default  
 88518 greater than or equal to 128 shall be used. The maximum number of commands in  
 88519 the history list is unspecified, but shall be at least 128. An implementation may  
 88520 choose to access this variable only when initializing the history file, as described  
 88521 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
 88522 after the history file has been initialized are effective.

88523 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 88524 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 88525 variables used to determine the values of locale categories.)

88526 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 88527 internationalization variables.

88528 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 88529 characters (for example, single-byte as opposed to multi-byte characters in  
 88530 arguments and input files).

88531 *LC\_MESSAGES*

88532 Determine the locale that should be used to affect the format and contents of  
 88533 diagnostic messages written to standard error.

- 88534 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 88535 **ASYNCHRONOUS EVENTS**
- 88536 Default.
- 88537 **STDOUT**
- 88538 When the **-l** option is used to list commands, the format of each command in the list shall be as follows:
- 88539
- 88540 "%d\t%s\n", <line number>, <command>
- 88541 If both the **-l** and **-n** options are specified, the format of each command shall be:
- 88542 "\t%s\n", <command>
- 88543 If the <command> consists of more than one line, the lines after the first shall be displayed as:
- 88544 "\t%s\n", <continued-command>
- 88545 **STDERR**
- 88546 The standard error shall be used only for diagnostic messages.
- 88547 **OUTPUT FILES**
- 88548 None.
- 88549 **EXTENDED DESCRIPTION**
- 88550 None.
- 88551 **EXIT STATUS**
- 88552 The following exit values shall be returned:
- 88553 0 Successful completion of the listing.
- 88554 >0 An error occurred.
- 88555 Otherwise, the exit status shall be that of the commands executed by *fc*.
- 88556 **CONSEQUENCES OF ERRORS**
- 88557 Default.
- 88558 **APPLICATION USAGE**
- 88559 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the *FCEDIT* editor, the command:
- 88560
- 88561 *fc -s | more*
- 88562
- 88563 does not work correctly on many systems.
- 88564 Users on windowing systems may want to have separate history files for each window by setting *HISTFILE* as follows:
- 88565
- 88566 *HISTFILE=\$HOME/.sh\_hist\$\$*
- 88567 **EXAMPLES**
- 88568 None.
- 88569 **RATIONALE**
- 88570 This utility is based on the *fc* built-in of the KornShell.
- 88571 An early proposal specified the **-e** option as [**-e** *editor* [*old= new* ]], which is not historical practice. Historical practice in *fc* of either [**-e** *editor*] or [**-e** - [*old= new* ]] is acceptable, but not both together. To clarify this, a new option **-s** was introduced replacing the [**-e** -]. This resolves

88574 the conflict and makes *fc* conform to the Utility Syntax Guidelines.

88575 *HISTFILE* Some implementations of the KornShell check for the superuser and do not create  
88576 a history file unless *HISTFILE* is set. This is done primarily to avoid creating  
88577 unlinked files in the root file system when logging in during single-user mode.  
88578 *HISTFILE* must be set for the superuser to have history.

88579 *HISTSIZE* Needed to limit the size of history files. It is the intent of the standard developers  
88580 that when two shells share the same history file, commands that are entered in one  
88581 shell shall be accessible by the other shell. Because of the difficulties of  
88582 synchronization over a network, the exact nature of the interaction is unspecified.

88583 The initialization process for the history file can be dependent on the system start-up files, in  
88584 that they may contain commands that effectively preempt the settings the user has for *HISTFILE*  
88585 and *HISTSIZE*. For example, function definition commands are recorded in the history file. If  
88586 the system administrator includes function definitions in some system start-up file called before  
88587 the *ENV* file, the history file is initialized before the user can influence its characteristics. In some  
88588 historical shells, the history file is initialized just after the *ENV* file has been processed. Because  
88589 of these situations, the text requires the initialization process to be implementation-defined.

88590 Consideration was given to omitting the *fc* utility in favor of the command line editing feature in  
88591 *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

88592 EDITOR=vi fc

88593 However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously  
88594 (such as *fc 10 20*) and to use editors other than those supported by *sh* for command line editing.

88595 In the KornShell, the alias *r* ("re-do") is preset to *fc -e -* (equivalent to the POSIX *fc -s*). This is  
88596 probably an easier command name to remember than *fc* ("fix command"), but it does not meet  
88597 the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this  
88598 description closely matches historical KornShell practice already, such a renaming was seen as  
88599 gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,  
88600 *grep*, or *yacc* are standardized by POSIX.

88601 Command numbers have no ordering effects; they are like serial numbers. The *-r* option and  
88602 *-number* operand address the sequence of command execution, regardless of serial numbers. So,  
88603 for example, if the command number wrapped back to 1 at some arbitrary point, there would be  
88604 no ambiguity associated with traversing the wrap point. For example, if the command history  
88605 were:

88606 32766: echo 1  
88607 32767: echo 2  
88608 1: echo 3

88609 the number *-2* refers to command 32767 because it is the second previous command, regardless  
88610 of serial number.

#### 88611 FUTURE DIRECTIONS

88612 None.

#### 88613 SEE ALSO

88614 *sh*

88615 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

88616 **CHANGE HISTORY**

88617 First released in Issue 4.

88618 **Issue 5**

88619 The FUTURE DIRECTIONS section is added.

88620 **Issue 6**

88621 This utility is marked as part of the User Portability Utilities option.

88622 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
88623 “directory referred to by the *HOME* environment variable”.88624 **Issue 7**

88625 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

88626 **NAME**

88627 fg — run jobs in the foreground

88628 **SYNOPSIS**88629 UP fg [*job\_id*]88630 **DESCRIPTION**88631 If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job  
88632 from the current environment (see Section 2.12, on page 2331) into the foreground.88633 Using *fg* to place a job into the foreground shall remove its process ID from the list of those  
88634 “known in the current shell execution environment”; see Section 2.9.3.1 (on page 2319).88635 **OPTIONS**

88636 None.

88637 **OPERANDS**

88638 The following operand shall be supported:

88639 *job\_id* Specify the job to be run as a foreground job. If no *job\_id* operand is given, the  
88640 *job\_id* for the job that was most recently suspended, placed in the background, or  
88641 run as a background job shall be used. The format of *job\_id* is described in XBD  
88642 Section 3.203 (on page 65).88643 **STDIN**

88644 Not used.

88645 **INPUT FILES**

88646 None.

88647 **ENVIRONMENT VARIABLES**88648 The following environment variables shall affect the execution of *fg*:88649 *LANG* Provide a default value for the internationalization variables that are unset or null.  
88650 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
88651 variables used to determine the values of locale categories.)88652 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
88653 internationalization variables.88654 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
88655 characters (for example, single-byte as opposed to multi-byte characters in  
88656 arguments).88657 *LC\_MESSAGES*88658 Determine the locale that should be used to affect the format and contents of  
88659 diagnostic messages written to standard error.88660 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.88661 **ASYNCHRONOUS EVENTS**

88662 Default.

88663 **STDOUT**88664 The *fg* utility shall write the command line of the job to standard output in the following format:88665 "%s\n", <*command*>

88666 **STDERR**

88667 The standard error shall be used only for diagnostic messages.

88668 **OUTPUT FILES**

88669 None.

88670 **EXTENDED DESCRIPTION**

88671 None.

88672 **EXIT STATUS**

88673 The following exit values shall be returned:

88674 0 Successful completion.

88675 &gt;0 An error occurred.

88676 **CONSEQUENCES OF ERRORS**88677 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the foreground.  
8867888679 **APPLICATION USAGE**88680 The *fg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell regular built-in.  
88681  
88682  
8868388684 **EXAMPLES**

88685 None.

88686 **RATIONALE**88687 The extensions to the shell specified in this volume of POSIX.1-2008 have mostly been based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of POSIX.1-2008 to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).  
88688  
88689  
88690  
88691  
88692  
8869388694 **FUTURE DIRECTIONS**

88695 None.

88696 **SEE ALSO**88697 [Section 2.9.3.1](#) (on page 2319), [Section 2.12](#) (on page 2331), *bg*, *kill*, *jobs*, *wait*88698 [XBD Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173)88699 **CHANGE HISTORY**

88700 First released in Issue 4.

88701 **Issue 6**

88702 This utility is marked as part of the User Portability Utilities option.

88703 The APPLICATION USAGE section is added.

88704 The JC marking is removed from the SYNOPSIS since job control is mandatory in this version.

88705 **NAME**

88706 file — determine file type

88707 **SYNOPSIS**

88708 file [-dh] [-M file] [-m file] file...

88709 file -i [-h] file...

88710 **DESCRIPTION**88711 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to  
88712 classify it:

- 88713 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output  
88714 shall indicate that the file was processed, but that its type could not be determined.
- 88715 2. If the file is not a regular file, its file type shall be identified. The file types directory,  
88716 FIFO, socket, block special, and character special shall be identified as such. Other  
88717 implementation-defined file types may also be identified. If *file* is a symbolic link, by  
88718 default the link shall be resolved and *file* shall test the type of file referenced by the  
88719 symbolic link. (See the **-h** and **-i** options below.)
- 88720 3. If the length of *file* is zero, it shall be identified as an empty file.
- 88721 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at  
88722 identifying its contents based on position-sensitive tests. (The answer is not guaranteed to  
88723 be correct; see the **-d**, **-M**, and **-m** options below.)
- 88724 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on  
88725 context-sensitive default system tests. (The answer is not guaranteed to be correct.)
- 88726 6. The file shall be identified as a data file.

88727 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall  
88728 indicate that the file was processed, but that its type could not be determined.88729 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file  
88730 referenced by the symbolic link.88731 **OPTIONS**88732 The *file* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **-m**,  
88733 **-d**, and **-M** options shall be significant.

88734 The following options shall be supported by the implementation:

- 88735 **-d** Apply any position-sensitive default system tests and context-sensitive default  
88736 system tests to the file. This is the default if no **-M** or **-m** option is specified.
- 88737 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is  
88738 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall  
88739 identify the file as a symbolic link, as if **-h** had been specified.
- 88740 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but  
88741 identify the file as specified in the STDOUT section.
- 88742 **-M file** Specify the name of a file containing position-sensitive tests that shall be applied to  
88743 a file in order to classify it (see the EXTENDED DESCRIPTION). No position-  
88744 sensitive default system tests nor context-sensitive default system tests shall be  
88745 applied unless the **-d** option is also specified.

88746 **-m file** Specify the name of a file containing position-sensitive tests that shall be applied to  
88747 a file in order to classify it (see the EXTENDED DESCRIPTION).

88748 If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-  
88749 sensitive default system tests shall be applied after the position-sensitive tests specified by the  
88750 **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**  
88751 option is specified with the **-d** option, the concatenation of the position-sensitive tests specified  
88752 by these options shall be applied in the order specified by the appearance of these options. If a  
88753 **-M** or **-m file** option-argument is **-**, the results are unspecified.

#### 88754 OPERANDS

88755 The following operand shall be supported:

88756 *file* A pathname of a file to be tested.

#### 88757 STDIN

88758 The standard input shall be used if a *file* operand is **'-'** and the implementation treats the **'-'**  
88759 as meaning standard input. Otherwise, the standard input shall not be used.

#### 88760 INPUT FILES

88761 The *file* can be any file type.

#### 88762 ENVIRONMENT VARIABLES

88763 The following environment variables shall affect the execution of *file*:

88764 *LANG* Provide a default value for the internationalization variables that are unset or null.  
88765 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
88766 variables used to determine the values of locale categories.)

88767 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
88768 internationalization variables.

88769 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
88770 characters (for example, single-byte as opposed to multi-byte characters in  
88771 arguments and input files).

88772 *LC\_MESSAGES*

88773 Determine the locale that should be used to affect the format and contents of  
88774 diagnostic messages written to standard error and informative messages written to  
88775 standard output.

88776 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 88777 ASYNCHRONOUS EVENTS

88778 Default

#### 88779 STDOUT

88780 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

88781 "%s: %s\n", <file>, <type>

88782 The values for <type> are unspecified, except that in the POSIX locale, if *file* is identified as one  
88783 of the types listed in the following table, <type> shall contain (but is not limited to) the  
88784 corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or  
88785 **-m** option. Each <space> shown in the strings shall be exactly one <space>.

88786

Table 4-9 File Utility Output Strings

|       | If <i>file</i> is:                                           | < <i>type</i> > shall contain the string: | Notes |
|-------|--------------------------------------------------------------|-------------------------------------------|-------|
| 88787 | Nonexistent                                                  | cannot open                               |       |
| 88788 | Block special                                                | block special                             | 1     |
| 88789 | Character special                                            | character special                         | 1     |
| 88790 | Directory                                                    | directory                                 | 1     |
| 88791 | FIFO                                                         | fifo                                      | 1     |
| 88792 | Socket                                                       | socket                                    | 1     |
| 88793 | Symbolic link                                                | symbolic link to                          | 1     |
| 88794 | Regular file                                                 | regular file                              | 1,2   |
| 88795 | Empty regular file                                           | empty                                     | 3     |
| 88796 | Regular file that cannot be read                             | cannot open                               | 3     |
| 88798 | Executable binary                                            | executable                                | 3,4,6 |
| 88799 | <i>ar</i> archive library (see <i>ar</i> )                   | archive                                   | 3,4,6 |
| 88800 | Extended <i>cpio</i> format (see <i>pax</i> )                | <i>cpio</i> archive                       | 3,4,6 |
| 88801 | Extended <i>tar</i> format (see <b>ustar</b> in <i>pax</i> ) | <i>tar</i> archive                        | 3,4,6 |
| 88802 | Shell script                                                 | commands text                             | 3,5,6 |
| 88803 | C-language source                                            | c program text                            | 3,5,6 |
| 88804 | FORTRAN source                                               | fortran program text                      | 3,5,6 |
| 88805 | Regular file whose type cannot be determined                 | data                                      | 3     |

88806 **Notes:**

- 88807 1. This is a file type test.
- 88808 2. This test is applied only if the **-i** option is specified.
- 88809 3. This test is applied only if the **-i** option is not specified.
- 88810 4. This is a position-sensitive default system test.
- 88811 5. This is a context-sensitive default system test.
- 88812 6. Position-sensitive default system tests and context-sensitive default system tests are not
- 88813 applied if the **-M** option is specified unless the **-d** option is also specified.

88814 In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following

88815 alternative output format shall be used:

88816 "%s: %s %s\n", <*file*>, <*type*>, <*contents of link*>"

88817 If the file named by the *file* operand does not exist, cannot be read, or the type of the file named

88818 by the *file* operand cannot be determined, this shall not be considered an error that affects the

88819 exit status.

88820 **STDERR**

88821 The standard error shall be used only for diagnostic messages.

88822 **OUTPUT FILES**

88823 None.

## 88824 EXTENDED DESCRIPTION

88825 A file specified as an option-argument to the `-m` or `-M` options shall contain one position-  
 88826 sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of  
 88827 the line shall be printed and no further tests shall be applied, with the exception that tests on  
 88828 immediately following lines beginning with a single `'>'` character shall be applied.

88829 Each line shall be composed of the following four <tab>-separated fields. (Implementations may  
 88830 allow any combination of one or more white-space characters other than <newline> to act as  
 88831 field separators.)

88832 *offset* An unsigned number (optionally preceded by a single `'>'` character) specifying  
 88833 the *offset*, in bytes, of the value in the file that is to be compared against the *value*  
 88834 field of the line. If the file is shorter than the specified offset, the test shall fail.

88835 If the *offset* begins with the character `'>'`, the test contained in the line shall not be  
 88836 applied to the file unless the test on the last line for which the *offset* did not begin  
 88837 with a `'>'` was successful. By default, the *offset* shall be interpreted as an unsigned  
 88838 decimal number. With a leading `0x` or `0X`, the *offset* shall be interpreted as a  
 88839 hexadecimal number; otherwise, with a leading `0`, the *offset* shall be interpreted as  
 88840 an octal number.

88841 *type* The type of the value in the file to be tested. The type shall consist of the type  
 88842 specification characters `d`, `s`, and `u`, specifying signed decimal, string, and  
 88843 unsigned decimal, respectively.

88844 The *type* string shall be interpreted as the bytes from the file starting at the  
 88845 specified *offset* and including the same number of bytes specified by the *value* field.  
 88846 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test  
 88847 shall fail.

88848 The type specification characters `d` and `u` can be followed by an optional unsigned  
 88849 decimal integer that specifies the number of bytes represented by the type. The  
 88850 type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`,  
 88851 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

88852 The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall  
 88853 correspond to their respective C-language types as follows. If the system claims  
 88854 conformance to the C-Language Development Utilities option, those specifiers  
 88855 shall correspond to the default sizes used in the `c99` utility. Otherwise, the default  
 88856 sizes shall be implementation-defined.

88857 For the type specifier characters `d` and `u`, the default number of bytes shall  
 88858 correspond to the size of a basic integer type of the implementation. For these  
 88859 specifier characters, the implementation shall support values of the optional  
 88860 number of bytes to be converted corresponding to the number of bytes in the C-  
 88861 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an  
 88862 application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when  
 88863 interpreting numeric values is implementation-defined, but shall correspond to the  
 88864 order in which a constant of the corresponding type is stored in memory on the  
 88865 system.

88866 All type specifiers, except for `s`, can be followed by a mask specifier of the form  
 88867 `&number`. The mask value shall be AND'ed with the value of the input file before  
 88868 the comparison with the *value* field of the line is made. By default, the mask shall  
 88869 be interpreted as an unsigned decimal number. With a leading `0x` or `0X`, the mask  
 88870 shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading

|       |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 88871 |                   | 0, the mask shall be interpreted as an unsigned octal number.                                                                                                                                                                                                                                                                                                                                          |
| 88872 |                   | The strings <b>byte</b> , <b>short</b> , <b>long</b> , and <b>string</b> shall also be supported as type fields, being interpreted as <i>dC</i> , <i>dS</i> , <i>dL</i> , and <i>s</i> , respectively.                                                                                                                                                                                                 |
| 88873 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88874 | <i>value</i>      | The <i>value</i> to be compared with the value from the file.                                                                                                                                                                                                                                                                                                                                          |
| 88875 |                   | If the specifier from the type field is <i>s</i> or <b>string</b> , then interpret the value as a string. Otherwise, interpret it as a number. If the value is a string, then the test shall succeed only when a string value exactly matches the bytes from the file.                                                                                                                                 |
| 88876 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88877 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88878 |                   | If the <i>value</i> is a string, it can contain the following sequences:                                                                                                                                                                                                                                                                                                                               |
| 88879 | <i>\character</i> | The <backslash>-escape sequences as specified in XBD Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequence '\ ' (the <backslash> character followed by a <space> character) shall be recognized to represent a <space> character. The results of using any other character, other than an octal digit, following the <backslash> are unspecified. |
| 88880 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88881 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88882 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88883 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88884 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88885 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88886 | <i>\octal</i>     | Octal sequences that can be used to represent characters with specific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three octal-digit characters (01234567).                                                                                                                                                                        |
| 88887 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88888 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88889 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88890 |                   | By default, any value that is not a string shall be interpreted as a signed decimal number. Any such value, with a leading 0x or 0X, shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading zero, the value shall be interpreted as an unsigned octal number.                                                                                                               |
| 88891 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88892 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88893 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88894 |                   | If the value is not a string, it can be preceded by a character indicating the comparison to be performed. Permissible characters and the comparisons they specify are as follows:                                                                                                                                                                                                                     |
| 88895 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88896 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88897 | =                 | The test shall succeed if the value from the file equals the <i>value</i> field.                                                                                                                                                                                                                                                                                                                       |
| 88898 | <                 | The test shall succeed if the value from the file is less than the <i>value</i> field.                                                                                                                                                                                                                                                                                                                 |
| 88899 | >                 | The test shall succeed if the value from the file is greater than the <i>value</i> field.                                                                                                                                                                                                                                                                                                              |
| 88900 | &                 | The test shall succeed if all of the set bits in the <i>value</i> field are set in the value from the file.                                                                                                                                                                                                                                                                                            |
| 88901 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88902 | ^                 | The test shall succeed if at least one of the set bits in the <i>value</i> field is not set in the value from the file.                                                                                                                                                                                                                                                                                |
| 88903 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88904 | x                 | The test shall succeed if the file is large enough to contain a value of the type specified starting at the offset specified.                                                                                                                                                                                                                                                                          |
| 88905 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88906 | <i>message</i>    | The <i>message</i> to be printed if the test succeeds. The <i>message</i> shall be interpreted using the notation for the <i>printf</i> formatting specification; see <i>printf</i> . If the <i>value</i> field was a string, then the value from the file shall be the argument for the <i>printf</i> formatting specification; otherwise, the value from the file shall be the argument.             |
| 88907 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88908 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |
| 88909 |                   |                                                                                                                                                                                                                                                                                                                                                                                                        |

88910 **EXIT STATUS**

88911 The following exit values shall be returned:

88912 0 Successful completion.

88913 &gt;0 An error occurred.

88914 **CONSEQUENCES OF ERRORS**

88915 Default.

88916 **APPLICATION USAGE**

88917 The *file* utility can only be required to guess at many of the file types because only exhaustive  
 88918 testing can determine some types with certainty. For example, binary data on some  
 88919 implementations might match the initial segment of an executable or a *tar* archive.

88920 Note that the table indicates that the output contains the stated string. Systems may add text  
 88921 before or after the string. For executables, as an example, the machine architecture and various  
 88922 facts about how the file was link-edited may be included. Note also that on systems that  
 88923 recognize shell script files starting with "#!" as executable files, these may be identified as  
 88924 executable binary files rather than as shell scripts.

88925 **EXAMPLES**

88926 Determine whether an argument is a binary executable file:

```
88927 file -- "$1" | grep -q '.*executable' &&  
88928 printf "%s is executable.\n" "$1"
```

88929 **RATIONALE**88930 The *-f* option was omitted because the same effect can (and should) be obtained using the *xargs*  
88931 utility.

88932 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,  
 88933 directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive  
 88934 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler  
 88935 source, *nroff/troff/eqn/tbl* source *troff* output, shell script, C shell script, English text, ASCII text,  
 88936 various executables, APL workspace, compiled terminfo entries, and CURSES screen images.  
 88937 Only those types that are reasonably well specified in POSIX or are directly related to POSIX  
 88938 utilities are listed in the table.

88939 Historical systems have used a "magic file" named */etc/magic* to help identify file types. Because  
 88940 it is generally useful for users and scripts to be able to identify special file types, the *-m* flag and  
 88941 a portable format for user-created magic files has been specified. No requirement is made that an  
 88942 implementation of *file* use this method of identifying files, only that users be permitted to add  
 88943 their own classifying tests.

88944 In addition, three options have been added to historical practice. The *-d* flag has been added to  
 88945 permit users to cause their tests to follow any default system tests. The *-i* flag has been added to  
 88946 permit users to test portably for regular files in shell scripts. The *-M* flag has been added to  
 88947 permit users to ignore any default system tests.

88948 The POSIX.1-2008 description of default system tests and the interaction between the *-d*, *-M*,  
 88949 and *-m* options did not clearly indicate that there were two types of "default system tests". The  
 88950 "position-sensitive tests" determine file types by looking for certain string or binary values at  
 88951 specific offsets in the file being examined. These position-sensitive tests were implemented in  
 88952 historical systems using the magic file described above. Some of these tests are now built into  
 88953 the *file* utility itself on some implementations so the output can provide more detail than can be  
 88954 provided by magic files. For example, a magic file can easily identify a *core* file on most  
 88955 implementations, but cannot name the program file that dropped the core. A magic file could

88956 produce output such as:

88957 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1

88958 but by building the test into the *file* utility, you could get output such as:

88959 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'

88960 These extended built-in tests are still to be treated as position-sensitive default system tests even  
88961 if they are not listed in */etc/magic* or any other magic file.

88962 The context-sensitive default system tests were always built into the *file* utility. These tests  
88963 looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and  
88964 other computer language source files, and even plain text files. With the addition of the *-m* and  
88965 *-M* options the distinction between position-sensitive and context-sensitive default system tests  
88966 became important because the order of testing is important. The context-sensitive system default  
88967 tests should never be applied before any position-sensitive tests even if the *-d* option is specified  
88968 before a *-m* option or *-M* option due to the high probability that the context-sensitive system  
88969 default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests  
88970 specified by the *-m* or *-M* option would be applied to give a more accurate identification.

88971 Leaving the meaning of *-M* – and *-m* – unspecified allows an existing prototype of these  
88972 options to continue to work in a backwards-compatible manner. (In that implementation, *-M* –  
88973 was roughly equivalent to *-d* in POSIX.1-2008.)

88974 The historical *-c* option was omitted as not particularly useful to users or portable shell scripts.  
88975 In addition, a reasonable implementation of the *file* utility would report any errors found each  
88976 time the magic file is read.

88977 The historical format of the magic file was the same as that specified by the Rationale in the  
88978 ISO POSIX-2:1993 standard for the *offset value*, and *message* fields; however, it used less precise  
88979 type fields than the format specified by the current normative text. The new type field values are  
88980 a superset of the historical ones.

88981 The following is an example magic file:

88982 0 short 070707 cpio archive  
88983 0 short 0143561 Byte-swapped cpio archive  
88984 0 string 070707 ASCII cpio archive  
88985 0 long 0177555 Very old archive  
88986 0 short 0177545 Old archive  
88987 0 short 017437 Old packed data  
88988 0 string \037\036 Packed data  
88989 0 string \377\037 Compacted data  
88990 0 string \037\235 Compressed data  
88991 >2 byte&0x80 >0 Block compressed  
88992 >2 byte&0x1f x %d bits  
88993 0 string \032\001 Compiled Terminfo Entry  
88994 0 short 0433 Curses screen image  
88995 0 short 0434 Curses screen image  
88996 0 string <ar> System V Release 1 archive  
88997 0 string !<arch>\n\_\_\_.SYMDEF Archive random library  
88998 0 string !<arch> Archive  
88999 0 string ARF\_BEGARF PHIGS clear text archive  
89000 0 long 0x137A2950 Scalable OpenFont binary  
89001 0 long 0x137A2951 Encrypted scalable OpenFont binary

- 89002 The use of a basic integer data type is intended to allow the implementation to choose a word  
89003 size commonly used by applications on that architecture.
- 89004 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
89005 but this has been modified in this version.
- 89006 **FUTURE DIRECTIONS**
- 89007 None.
- 89008 **SEE ALSO**
- 89009 *ar, ls, pax, printf*
- 89010 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 89011 **CHANGE HISTORY**
- 89012 First released in Issue 4.
- 89013 **Issue 6**
- 89014 This utility is marked as part of the User Portability Utilities option.
- 89015 Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft  
89016 standard.
- 89017 IEEE PASC Interpretations 1003.2 #192 and #178 are applied.
- 89018 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to  
89019 address ambiguities raised in defect reports.
- 89020 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the  
89021 OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the  
89022 Utility Syntax Guidelines.
- 89023 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification  
89024 characters.
- 89025 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application  
89026 developers to create portable magic files that can match characters in strings, and allowing  
89027 common extensions found in existing implementations.
- 89028 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing  
89029 behavior on systems with bytes consisting of more than eight bits.
- 89030 **Issue 7**
- 89031 Austin Group Interpretation 1003.1-2001 #092 is applied.
- 89032 SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in [Table 4-9](#).
- 89033 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 89034 The *file* utility is moved from the User Portability Utilities option to the Base. User Portability  
89035 Utilities is now an option for interactive utilities.
- 89036 The EXAMPLES section is revised to correct an error with the pathname "`$1`".

89037 **NAME**

89038 find — find files

89039 **SYNOPSIS**89040 find [-H|-L] *path...* [*operand\_expression...*]89041 **DESCRIPTION**

89042 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,  
 89043 evaluating a Boolean expression composed of the primaries described in the OPERANDS section  
 89044 for each file encountered. Each *path* operand shall be evaluated unaltered as it was provided,  
 89045 including all trailing <slash> characters; all pathnames for other files encountered in the  
 89046 hierarchy shall consist of the concatenation of the current *path* operand, a <slash> if the current  
 89047 *path* operand did not end in one, and the filename relative to the *path* operand. The relative  
 89048 portion shall contain no dot or dot-dot components, no trailing <slash> characters, and only  
 89049 single <slash> characters between pathname components.

89050 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail  
 89051 due to path length limitations (unless a *path* operand specified by the application exceeds  
 89052 {PATH\_MAX} requirements).

89053 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 89054 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a  
 89055 diagnostic message to standard error and shall either recover its position in the hierarchy or  
 89056 terminate.

89057 **OPTIONS**89058 The *find* utility shall conform to XBD Section 12.2 (on page 215).

89059 The following options shall be supported by the implementation:

89060 **-H** Cause the file information and file type evaluated for each symbolic link  
 89061 encountered as a *path* operand on the command line to be those of the file  
 89062 referenced by the link, and not the link itself. If the referenced file does not exist,  
 89063 the file information and type shall be for the link itself. File information and type  
 89064 for symbolic links encountered during the traversal of a file hierarchy shall be that  
 89065 of the link itself.

89066 **-L** Cause the file information and file type evaluated for each symbolic link  
 89067 encountered as a *path* operand on the command line or encountered during the  
 89068 traversal of a file hierarchy to be those of the file referenced by the link, and not the  
 89069 link itself. If the referenced file does not exist, the file information and type shall be  
 89070 for the link itself.

89071 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 89072 an error. The last option specified shall determine the behavior of the utility. If neither the **-H**  
 89073 nor the **-L** option is specified, then the file information and type for symbolic links encountered  
 89074 as a *path* operand on the command line or encountered during the traversal of a file hierarchy  
 89075 shall be that of the link itself.

89076 **OPERANDS**

89077 The following operands shall be supported:

89078 The first operand and subsequent operands up to but not including the first operand that starts  
 89079 with a '-', or is a '!' or a '(', shall be interpreted as *path* operands. If the first operand starts  
 89080 with a '-', or is a '!' or a '(', the behavior is unspecified. Each *path* operand is a pathname of  
 89081 a starting point in the file hierarchy.

89082 The first operand that starts with a '-', or is a '!' or a '(', and all subsequent arguments shall

**find**

89083 be interpreted as an *expression* made up of the following primaries and operators. In the  
 89084 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal  
 89085 integer optionally preceded by a plus ('+') or minus-sign ('-') sign, as follows:

89086 **+n** More than *n*.

89087 **n** Exactly *n*.

89088 **-n** Less than *n*.

89089 The following primaries shall be supported:

89090 **-name** *pattern*

89091 The primary shall evaluate as true if the basename of the current pathname  
 89092 matches *pattern* using the pattern matching notation described in Section 2.13 (on  
 89093 page 2332). The additional rules in Section 2.13.3 (on page 2333) do not apply as  
 89094 this is a matching operation, not an expansion.

89095 **-path** *pattern*

89096 The primary shall evaluate as true if the current pathname matches *pattern* using  
 89097 the pattern matching notation described in Section 2.13 (on page 2332). The  
 89098 additional rules in Section 2.13.3 (on page 2333) do not apply as this is a matching  
 89099 operation, not an expansion.

89100 **-nouser**

89101 The primary shall evaluate as true if the file belongs to a user ID for which the  
 89102 *getpwuid()* function defined in the System Interfaces volume of POSIX.1-2008 (or  
 equivalent) returns NULL.

89103 **-nogroup**

89104 The primary shall evaluate as true if the file belongs to a group ID for which the  
 89105 *getgrgid()* function defined in the System Interfaces volume of POSIX.1-2008 (or  
 equivalent) returns NULL.

89106 **-xdev**

89107 The primary shall always evaluate as true; it shall cause *find* not to continue  
 89108 descending past directories that have a different device ID (*st\_dev*, see the *stat()*  
 89109 function defined in the System Interfaces volume of POSIX.1-2008). If any **-xdev**  
 89110 primary is specified, it shall apply to the entire expression even if the **-xdev**  
 primary would not normally be evaluated.

89111 **-prune**

89112 The primary shall always evaluate as true; it shall cause *find* not to descend the  
 89113 current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**  
 primary shall have no effect.

89114 **-perm** [-]*mode*

89115 The *mode* argument is used to represent file mode bits. It shall be identical in  
 89116 format to the *symbolic\_mode* operand described in *chmod*, and shall be interpreted  
 89117 as follows. To start, a template shall be assumed with all file mode bits cleared. An  
 89118 *op* symbol of '+' shall set the appropriate mode bits in the template; '-' shall  
 89119 clear the appropriate bits; '=' shall set the appropriate mode bits, without regard  
 89120 to the contents of the file mode creation mask of the process. The *op* symbol of '-'  
 89121 cannot be the first character of *mode*; this avoids ambiguity with the optional  
 89122 leading <hyphen>. Since the initial mode is all bits off, there are not any symbolic  
 89123 modes that need to use '-' as the first character.

89124 If the <hyphen> is omitted, the primary shall evaluate as true when the file  
 89125 permission bits exactly match the value of the resulting template.

89126 Otherwise, if *mode* is prefixed by a <hyphen>, the primary shall evaluate as true if  
 89127 at least all the bits in the resulting template are set in the file permission bits.

- 89128        **-perm** [-] *onum*
- 89129            If the <hyphen> is omitted, the primary shall evaluate as true when the file mode bits exactly match the value of the octal number *onum* (see the description of the octal *mode* in *chmod*). Otherwise, if *onum* is prefixed by a <hyphen>, the primary shall evaluate as true if at least all of the bits specified in *onum* are set. In both cases, the behavior is unspecified when *onum* exceeds 07777.
- 89130
- 89131
- 89132
- 89133
- 89134        **-type** *c*
- 89135            The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c', 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory, symbolic link, FIFO, regular file, or socket, respectively.
- 89136
- 89137        **-links** *n*
- 89138            The primary shall evaluate as true if the file has *n* links.
- 89139
- 89140        **-user** *uname*
- 89141            The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is a decimal integer and the *getpwnam()* (or equivalent) function does not return a valid user name, *uname* shall be interpreted as a user ID.
- 89142
- 89143
- 89144        **-group** *gname*
- 89145            The primary shall evaluate as true if the file belongs to the group *gname*. If *gname* is a decimal integer and the *getgrnam()* (or equivalent) function does not return a valid group name, *gname* shall be interpreted as a group ID.
- 89146
- 89147        **-size** *n*[*c*]
- 89148            The primary shall evaluate as true if the file size in bytes, divided by 512 and rounded up to the next integer, is *n*. If *n* is followed by the character '*c*', the size shall be in bytes.
- 89149
- 89148        **-atime** *n*
- 89149            The primary shall evaluate as true if the file access time subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 89150
- 89151        **-ctime** *n*
- 89152            The primary shall evaluate as true if the time of last change of file status information subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 89153
- 89154        **-mtime** *n*
- 89155            The primary shall evaluate as true if the file modification time subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 89155        **-exec** *utility\_name* [*argument* ...];
- 89156        **-exec** *utility\_name* [*argument* ...] {} +
- 89157            The end of the primary expression shall be punctuated by a <semicolon> or by a <plus-sign>. Only a <plus-sign> that immediately follows an argument containing the two characters "{}" shall punctuate the end of the primary expression. Other uses of the <plus-sign> shall not be treated as special.
- 89158
- 89159
- 89160
- 89161            If the primary expression is punctuated by a <semicolon>, the utility *utility\_name* shall be invoked once for each pathname and the primary shall evaluate as true if the utility returns a zero value as exit status. A *utility\_name* or *argument* containing only the two characters "{}" shall be replaced by the current pathname.
- 89162
- 89163
- 89164
- 89165            If the primary expression is punctuated by a <plus-sign>, the primary shall always evaluate as true, and the pathnames for which the primary is evaluated shall be aggregated into sets. The utility *utility\_name* shall be invoked once for each set of aggregated pathnames. Each invocation shall begin after the last pathname in the set is aggregated, and shall be completed before the *find* utility exits and before the first pathname in the next set (if any) is aggregated for this primary, but it is otherwise unspecified whether the invocation occurs before, during, or after the evaluations of other primaries. If any invocation returns a non-zero value as exit status, the *find* utility shall return a non-zero exit status. An argument containing
- 89166
- 89167
- 89168
- 89169
- 89170
- 89171
- 89172
- 89173

## find

## Utilities

- 89174 only the two characters "{}" shall be replaced by the set of aggregated  
 89175 pathnames, with each pathname passed as a separate argument to the invoked  
 89176 utility in the same order that it was aggregated. The size of any set of two or more  
 89177 pathnames shall be limited such that execution of the utility does not cause the  
 89178 system's {ARG\_MAX} limit to be exceeded. If more than one argument containing  
 89179 only the two characters "{}" is present, the behavior is unspecified.
- 89180 If a *utility\_name* or *argument* string contains the two characters "{}", but not just  
 89181 the two characters "{}", it is implementation-defined whether *find* replaces those  
 89182 two characters or uses the string without change. The current directory for the  
 89183 invocation of *utility\_name* shall be the same as the current directory when the *find*  
 89184 utility was started. If the *utility\_name* names any of the special built-in utilities (see  
 89185 [Section 2.14](#), on page 2334), the results are undefined.
- 89186 **-ok** *utility\_name* [*argument* ...];  
 89187 The **-ok** primary shall be equivalent to **-exec**, except that the use of a <plus-sign>  
 89188 to punctuate the end of the primary expression need not be supported, and *find*  
 89189 shall request affirmation of the invocation of *utility\_name* using the current file as  
 89190 an argument by writing to standard error as described in the STDERR section. If  
 89191 the response on standard input is affirmative, the utility shall be invoked.  
 89192 Otherwise, the command shall not be invoked and the value of the **-ok** operand  
 89193 shall be false.
- 89194 **-print** The primary shall always evaluate as true; it shall cause the current pathname to  
 89195 be written to standard output.
- 89196 **-newer** *file* The primary shall evaluate as true if the modification time of the current file is  
 89197 more recent than the modification time of the file named by the pathname *file*.
- 89198 **-depth** The primary shall always evaluate as true; it shall cause descent of the directory  
 89199 hierarchy to be done so that all entries in a directory are acted on before the  
 89200 directory itself. If a **-depth** primary is not specified, all entries in a directory shall  
 89201 be acted on after the directory itself. If any **-depth** primary is specified, it shall  
 89202 apply to the entire expression even if the **-depth** primary would not normally be  
 89203 evaluated.
- 89204 The primaries can be combined using the following operators (in order of decreasing  
 89205 precedence):
- 89206 (*expression*) True if *expression* is true.
- 89207 ! *expression* Negation of a primary; the unary NOT operator.
- 89208 *expression* [-a] *expression*  
 89209 Conjunction of primaries; the AND operator is implied by the juxtaposition of two  
 89210 primaries or made explicit by the optional **-a** operator. The second expression shall  
 89211 not be evaluated if the first expression is false.
- 89212 *expression* -o *expression*  
 89213 Alternation of primaries; the OR operator. The second expression shall not be  
 89214 evaluated if the first expression is true.
- 89215 If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given  
 89216 expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression  
 89217 shall be effectively replaced by:
- 89218 ( *given\_expression* ) -print

- 89219 The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only  
89220 once.
- 89221 When the file type evaluated for the current file is a symbolic link, the results of evaluating the  
89222 **-perm** primary are implementation-defined.
- 89223 **STDIN**
- 89224 If the **-ok** primary is used, the response shall be read from the standard input. An entire line  
89225 shall be read as the response. Otherwise, the standard input shall not be used.
- 89226 **INPUT FILES**
- 89227 None.
- 89228 **ENVIRONMENT VARIABLES**
- 89229 The following environment variables shall affect the execution of *find*:
- 89230 **LANG** Provide a default value for the internationalization variables that are unset or null.  
89231 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
89232 variables used to determine the values of locale categories.)
- 89233 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
89234 internationalization variables.
- 89235 **LC\_COLLATE**
- 89236 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
89237 character collating elements used in the pattern matching notation for the **-n**  
89238 option and in the extended regular expression defined for the **yesexpr** locale  
89239 keyword in the *LC\_MESSAGES* category.
- 89240 **LC\_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of  
89241 text data as characters (for example, single-byte as opposed to multi-byte  
89242 characters in arguments), the behavior of character classes within the pattern  
89243 matching notation used for the **-n** option, and the behavior of character classes  
89244 within regular expressions used in the extended regular expression defined for the  
89245 **yesexpr** locale keyword in the *LC\_MESSAGES* category.
- 89246 **LC\_MESSAGES**
- 89247 Determine the locale used to process affirmative responses, and the locale used to  
89248 affect the format and contents of diagnostic messages and prompts written to  
89249 standard error.
- 89250 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 89251 **PATH** Determine the location of the *utility\_name* for the **-exec** and **-ok** primaries, as  
89252 described in XBD Chapter 8 (on page 173).
- 89253 **ASYNCHRONOUS EVENTS**
- 89254 Default.
- 89255 **STDOUT**
- 89256 The **-print** primary shall cause the current pathnames to be written to standard output. The  
89257 format shall be:
- 89258 "%s\n", <path>

**find**89259 **STDERR**

89260 The **-ok** primary shall write a prompt to standard error containing at least the *utility\_name* to be  
 89261 invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall  
 89262 be ' ? '. The exact format used is unspecified.

89263 Otherwise, the standard error shall be used only for diagnostic messages.

89264 **OUTPUT FILES**

89265 None.

89266 **EXTENDED DESCRIPTION**

89267 None.

89268 **EXIT STATUS**

89269 The following exit values shall be returned:

89270 0 All *path* operands were traversed successfully.

89271 >0 An error occurred.

89272 **CONSEQUENCES OF ERRORS**

89273 Default.

89274 **APPLICATION USAGE**

89275 When used in operands, pattern matching notation, <semicolon>, <left-parenthesis>, and  
 89276 <right-parenthesis> characters are special to the shell and must be quoted (see [Section 2.2](#), on  
 89277 page 2298).

89278 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary  
 89279 using the octal number argument form. Since this bit is not defined by this volume of  
 89280 POSIX.1-2008, applications must not assume that it actually refers to the traditional sticky bit.

89281 **EXAMPLES**

89282 1. The following commands are equivalent:

```
89283 find .
89284 find . -print
```

89285 They both write out the entire directory hierarchy from the current directory.

89286 2. The following command:

```
89287 find / \( -name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;
```

89288 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or  
 89289 more 24-hour periods.

89290 3. The following command:

```
89291 find . -perm -o+w,+s
```

89292 prints (**-print** is assumed) the names of all files in or below the current directory, with all  
 89293 of the file permission bits **S\_ISUID**, **S\_ISGID**, and **S\_IWOTH** set.

89294 4. The following command:

```
89295 find . -name SCCS -prune -o -print
```

89296 recursively prints pathnames of all files in the current directory and below, but skips  
 89297 directories named **SCCS** and files in them.

- 89298 5. The following command:
- 89299 `find . -print -name SCCS -prune`
- 89300 behaves as in the previous example, but prints the names of the SCCS directories.
- 89301 6. The following command is roughly equivalent to the `-nt` extension to `test`:
- 89302 `if [ -n "$(find file1 -prune -newer file2)" ]; then`
- 89303 `printf %s\n "file1 is newer than file2"`
- 89304 `fi`
- 89305 7. The descriptions of `-atime`, `-ctime`, and `-mtime` use the terminology  $n$  “86400 second
- 89306 periods (days)”. For example, a file accessed at 23:59 is selected by:
- 89307 `find . -atime -1 -print`
- 89308 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight
- 89309 boundary between days has no effect on the 24-hour calculation.
- 89310 8. The following command:
- 89311 `find . ! -name . -prune -name '*.old' -exec \`
- 89312 `sh -c 'mv "$@" ../old/' sh {} +`
- 89313 performs the same task as:
- 89314 `mv /*.old /*.old ../old/`
- 89315 while avoiding an “Argument list too long” error if there are a large number of files
- 89316 ending with `.old` (and avoiding “No such file or directory” errors if no files match `/*.old`
- 89317 or `/*.old`).
- 89318 The alternative:
- 89319 `find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;`
- 89320 is less efficient if there are many files to move because it executes one `mv` command per
- 89321 file.
- 89322 9. On systems configured to mount removable media on directories under `/media`, the
- 89323 following command searches the file hierarchy for files larger than 100 000 KB without
- 89324 searching any mounted removable media:
- 89325 `find -path /media -prune -o -size +200000 -print`
- 89326 10. Except for the root directory, and `"/"` on implementations where `"/"` does not refer to
- 89327 the root directory, no pattern given to `-name` will match a `<slash>`, because trailing
- 89328 `<slash>` characters are ignored when computing the basename of the file under
- 89329 evaluation. Given two empty directories named `foo` and `bar`, the following command:
- 89330 `find foo/// bar/// -name foo -o -name 'bar?*'`
- 89331 prints only the line `foo///`.

**RATIONALE**

89332 The `-a` operator was retained as an optional operator for compatibility with historical shell

89333 scripts, even though it is redundant with expression concatenation.

89334

89335 The descriptions of the `'-'` modifier on the `mode` and `onum` arguments to the `-perm` primary

89336 agree with historical practice on BSD and System V implementations. System V and BSD

89337 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,

89338 but checks for having at least all of the matching bits set instead of having exactly the matching

- 89339 bits set.
- 89340 The exact format of the interactive prompts is unspecified. Only the general nature of the  
89341 contents of prompts are specified because:
- 89342 • Implementations may desire more descriptive prompts than those used on historical  
89343 implementations.
  - 89344 • Since the historical prompt strings do not terminate with <newline> characters, there is no  
89345 portable way for another program to interact with the prompts of this utility via pipes.
- 89346 Therefore, an application using this prompting option relies on the system to provide the most  
89347 suitable dialog directly with the user, based on the general guidelines specified.
- 89348 The **-name** *file* operand was changed to use the shell pattern matching notation so that *find* is  
89349 consistent with other utilities using pattern matching.
- 89350 The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in  
89351 the file system. The intent is that the *st\_size* field defined in the System Interfaces volume of  
89352 POSIX.1-2008 should be used, not the *st\_blocks* found in historical implementations. There are at  
89353 least two reasons for this:
- 89354 1. In both System V and BSD, *find* only uses *st\_size* in size calculations for the operands  
89355 specified by this volume of POSIX.1-2008. (BSD uses *st\_blocks* only when processing the  
89356 **-ls** primary.)
  - 89357 2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls*  
89358 utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st\_size* for the  
89359 **-l** option size field and uses *st\_blocks* for the *ls -s* calculations. This volume of  
89360 POSIX.1-2008 does not specify *ls -s*.)
- 89361 The descriptions of **-atime**, **-ctime**, and **-mtime** were changed from the SVID description of *n*  
89362 "days" to *n* being the result of the integer division of the time difference in seconds by 86 400.  
89363 The description is also different in terms of the exact timeframe for the *n* case (*versus* the *+n* or  
89364 *-n*), but it matches all known historical implementations. It refers to one 86 400 second period in  
89365 the past, not any time from the beginning of that period to the current time. For example, **-atime**  
89366 2 is true if the file was accessed any time in the period from 72 hours to 48 hours ago.
- 89367 Historical implementations do not modify "{}" when it appears as a substring of an **-exec** or  
89368 **-ok** *utility\_name* or argument string. There have been numerous user requests for this extension,  
89369 so this volume of POSIX.1-2008 allows the desired behavior. At least one recent implementation  
89370 does support this feature, but encountered several problems in managing memory allocation  
89371 and dealing with multiple occurrences of "{}" in a string while it was being developed, so it is  
89372 not yet required behavior.
- 89373 Assuming the presence of **-print** was added to correct a historical pitfall that plagues novice  
89374 users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest  
89375 form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers  
89376 agreed that adding **-print** as a default expression was the correct decision and have added the  
89377 fast *find* functionality within a new utility called *locate*.
- 89378 Historically, the **-L** option was implemented using the primary **-follow**. The **-H** and **-L** options  
89379 were added for two reasons. First, they offer a finer granularity of control and consistency with  
89380 other programs that walk file hierarchies. Second, the **-follow** primary always evaluated to true.  
89381 As they were historically really global variables that took effect before the traversal began, some  
89382 valid expressions had unexpected results. An example is the expression **-print -o -follow**.  
89383 Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow**  
89384 would never be evaluated. This was never the case. Historical practice for the **-follow** primary,

89385 however, is not consistent. Some implementations always follow symbolic links on the  
 89386 command line whether **-follow** is specified or not. Others follow symbolic links on the  
 89387 command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L**  
 89388 options, but scripts using the current **-follow** primary would be broken if the **-follow** option is  
 89389 specified to work either way.

89390 Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links  
 89391 that still exist after symbolic links have been resolved, the command:

```
89392 find -L . -type l
```

89393 prints a list of symbolic links reachable from the current directory that do not resolve to  
 89394 accessible files.

89395 A feature of SVR4's *find* utility was the **-exec** primary's **+** terminator. This allowed filenames  
 89396 containing special characters (especially <newline> characters) to be grouped together without  
 89397 the problems that occur if such filenames are piped to *xargs*. Other implementations have added  
 89398 other ways to get around this problem, notably a **-print0** primary that wrote filenames with a  
 89399 null byte terminator. This was considered here, but not adopted. Using a null terminator meant  
 89400 that any utility that was going to process *find*'s **-print0** output had to add a new option to parse  
 89401 the null terminators it would now be reading.

89402 The "**-exec** ... { } +" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210.  
 89403 It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For  
 89404 example, the following command prints all files with a '-' after their name if they are regular  
 89405 files, and a '+' otherwise:

```
89406 find / -type f -exec echo {} - ';' \; -o -exec echo {} + ';' \;
```

89407 The change invalidates usage like this. Even though the previous standard stated that this usage  
 89408 would work, in practice many did not support it and the standard developers felt it better to  
 89409 now state that this was not allowable.

#### 89410 FUTURE DIRECTIONS

89411 None.

#### 89412 SEE ALSO

89413 [Section 2.2](#) (on page 2298), [Section 2.13](#) (on page 2332), [Section 2.14](#) (on page 2334), *chmod*, *pax*,  
 89414 *sh*, *test*

89415 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

89416 XSH *fstatat()*, *getgrgid()*, *getpwuid()*

#### 89417 CHANGE HISTORY

89418 First released in Issue 2.

#### 89419 Issue 5

89420 The FUTURE DIRECTIONS section is added.

#### 89421 Issue 6

89422 The following new requirements on POSIX implementations derive from alignment with the  
 89423 Single UNIX Specification:

- 89424 • The **-perm [-]onum** primary is supported.

89425 The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of  
 89426 symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

89427 IEEE PASC Interpretation 1003.2 #210 is applied, extending the **-exec** operand.

- 89428 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE  
89429 section to be consistent with the normative text.
- 89430 **Issue 7**
- 89431 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
89432 *LC\_MESSAGES* environment variable.
- 89433 Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the **-exec**  
89434 primary to be “immediately follows”.
- 89435 Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the **-H**  
89436 and **-L** options.
- 89437 Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the  
89438 evaluation of *path* operands.
- 89439 Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first  
89440 operand.
- 89441 SD5-XCU-ERN-48 is applied, clarifying the **-L** option in the case that the referenced file does not  
89442 exist.
- 89443 SD5-XCU-ERN-89 is applied, updating the OPERANDS section.
- 89444 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 89445 SD5-XCU-ERN-117 is applied, clarifying the **-perm** operand.
- 89446 SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.
- 89447 The description of the **-name** primary is revised and the **-path** primary is added (with a new  
89448 example).

89449 **NAME**

89450 fold — filter for folding lines

89451 **SYNOPSIS**89452 fold [-bs] [-w *width*] [*file...*]89453 **DESCRIPTION**

89454 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a  
 89455 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be  
 89456 broken by the insertion of a <newline> such that each output line (referred to later in this section  
 89457 as a *segment*) is the maximum width possible that does not exceed the specified number of  
 89458 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior  
 89459 is undefined if *width* is less than the number of columns any single character in the input would  
 89460 occupy.

89461 If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the  
 89462 **-b** option is not specified, they shall be treated specially:

89463 <backspace> The current count of line width shall be decremented by one, although the count  
 89464 never shall become negative. The *fold* utility shall not insert a <newline>  
 89465 immediately before or after any <backspace>, unless the following character has a  
 89466 width greater than 1 and would cause the line width to exceed *width*.

89467 <carriage-return>

89468 The current count of line width shall be set to zero. The *fold* utility shall not insert a  
 89469 <newline> immediately before or after any <carriage-return>.

89470 <tab> Each <tab> encountered shall advance the column position pointer to the next tab  
 89471 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

89472 **OPTIONS**89473 The *fold* utility shall conform to XBD Section 12.2 (on page 215).

89474 The following options shall be supported:

89475 **-b** Count *width* in bytes rather than column positions.

89476 **-s** If a segment of a line contains a <blank> within the first *width* column positions (or  
 89477 bytes), break the line after the last such <blank> meeting the width constraints. If  
 89478 there is no <blank> meeting the requirements, the **-s** option shall have no effect for  
 89479 that output segment of the input line.

89480 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).  
 89481 The results are unspecified if *width* is not a positive decimal number. The default  
 89482 value shall be 80.

89483 **OPERANDS**

89484 The following operand shall be supported:

89485 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard  
 89486 input shall be used.

89487 **STDIN**

89488 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 89489 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 89490 the standard input shall not be used. See the INPUT FILES section.

**fold**

Utilities

89491 **INPUT FILES**

89492 If the **-b** option is specified, the input files shall be text files except that the lines are not limited  
 89493 to {LINE\_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

89494 **ENVIRONMENT VARIABLES**

89495 The following environment variables shall affect the execution of *fold*:

89496 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 89497 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 89498 variables used to determine the values of locale categories.)

89499 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 89500 internationalization variables.

89501 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 89502 characters (for example, single-byte as opposed to multi-byte characters in  
 89503 arguments and input files), and for the determination of the width in column  
 89504 positions each character would occupy on a constant-width font output device.

89505 **LC\_MESSAGES**

89506 Determine the locale that should be used to affect the format and contents of  
 89507 diagnostic messages written to standard error.

89508 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

89509 **ASYNCHRONOUS EVENTS**

89510 Default.

89511 **STDOUT**

89512 The standard output shall be a file containing a sequence of characters whose order shall be  
 89513 preserved from the input files, possibly with inserted <newline> characters.

89514 **STDERR**

89515 The standard error shall be used only for diagnostic messages.

89516 **OUTPUT FILES**

89517 None.

89518 **EXTENDED DESCRIPTION**

89519 None.

89520 **EXIT STATUS**

89521 The following exit values shall be returned:

89522 0 All input files were processed successfully.

89523 >0 An error occurred.

89524 **CONSEQUENCES OF ERRORS**

89525 Default.

89526 **APPLICATION USAGE**

89527 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.  
 89528 The *cut* utility should be used when the number of lines (or records) needs to remain constant.  
 89529 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

89530 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines  
 89531 wider than the printer is able to print (usually 80 or 132 column positions).

89532 **EXAMPLES**

89533 An example invocation that submits a file of possibly long lines to the printer (under the  
 89534 assumption that the user knows the line width of the printer to be assigned by *lp*):

89535 `fold -w 132 bigfile | lp`

89536 **RATIONALE**

89537 Although terminal input in canonical processing mode requires the erase character (frequently  
 89538 set to <backspace>) to erase the previous character (not byte or column position), terminal  
 89539 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the  
 89540 interpretation depends entirely on the physical device that actually displays/prints/stores the  
 89541 output. In all known internationalized implementations, the utilities producing output for  
 89542 mixed column-width output assume that a <backspace> character backs up one column position  
 89543 and outputs enough <backspace> characters to return to the start of the character when  
 89544 <backspace> is used to provide local line motions to support underlining and boldening  
 89545 operations. Since *fold* without the *-b* option is dealing with these same constraints, <backspace>  
 89546 is always treated as backing up one column position rather than backing up one character.

89547 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column  
 89548 position when written out. This is no longer always true. Since the most common usage of *fold* is  
 89549 believed to be folding long lines for output to limited-length output devices, this capability was  
 89550 preserved as the default case. The *-b* option was added so that applications could *fold* files with  
 89551 arbitrary length lines into text files that could then be processed by the standard utilities. Note  
 89552 that although the width for the *-b* option is in bytes, a line is never split in the middle of a  
 89553 character. (It is unspecified what happens if a width is specified that is too small to hold a single  
 89554 character found in the input followed by a <newline>.)

89555 The tab stops are hardcoded to be every eighth column to meet historical practice. No new  
 89556 method of specifying other tab stops was invented.

89557 **FUTURE DIRECTIONS**

89558 None.

89559 **SEE ALSO**

89560 *cut*

89561 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

89562 **CHANGE HISTORY**

89563 First released in Issue 4.

89564 **Issue 6**

89565 The normative text is reworded to avoid use of the term “must” for application requirements.

89566 **Issue 7**

89567 Austin Group Interpretation 1003.1-2001 #092 is applied.

**fold**

- 89568 Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify when a <newline> can be inserted before or after a <backspace>.
- 89569
- 89570 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

89571 **NAME**89572 fort77 — FORTRAN compiler (**FORTRAN**)89573 **SYNOPSIS**

```
89574 FD fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s]
89575 [-w] operand...
```

89576 **DESCRIPTION**

89577 The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full  
 89578 FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually  
 89579 consists of a compiler and link editor. The files referenced by *operands* are compiled and linked  
 89580 to produce an executable file. It is unspecified whether the linking occurs entirely within the  
 89581 operation of *fort77*; some implementations may produce objects that are not fully resolved until  
 89582 the file is executed.

89583 If the `-c` option is present, for all pathname operands of the form *file.f*, the files:

89584 `$(basename pathname.f) .o`

89585 shall be created or overwritten as the result of successful compilation. If the `-c` option is not  
 89586 specified, it is unspecified whether such `.o` files are created or deleted for the *file.f* operands.

89587 If there are no options that prevent link editing (such as `-c`) and all operands compile and link  
 89588 without error, the resulting executable file shall be written into the file named by the `-o` option  
 89589 (if present) or to the file **a.out**. The executable file shall be created as specified in the System  
 89590 Interfaces volume of POSIX.1-2008, except that the file permissions shall be set to:

89591 `S_IRWXO | S_IRWXG | S_IRWXU`

89592 and that the bits specified by the *umask* of the process shall be cleared.

89593 **OPTIONS**

89594 The *fort77* utility shall conform to XBD Section 12.2 (on page 215), except that:

- 89595 • The `-l library` operands have the format of options, but their position within a list of  
 89596 operands affects the order in which libraries are searched.
- 89597 • The order of specifying the multiple `-L` options is significant.
- 89598 • Conforming applications shall specify each option separately; that is, grouping option  
 89599 letters (for example, `-cg`) need not be recognized by all implementations.

89600 The following options shall be supported:

- 89601 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files  
 89602 that are produced.
- 89603 `-g` Produce symbolic information in the object or executable files; the nature of this  
 89604 information is unspecified, and may be modified by implementation-defined  
 89605 interactions with other options.
- 89606 `-s` Produce object or executable files, or both, from which symbolic and other  
 89607 information not required for proper execution using the *exec* family of functions  
 89608 defined in the System Interfaces volume of POSIX.1-2008 has been removed  
 89609 (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.
- 89610 `-o outfile` Use the pathname *outfile*, instead of the default **a.out**, for the executable file  
 89611 produced. If the `-o` option is present with `-c`, the result is unspecified.

- 89612        **-L** *directory*    Change the algorithm of searching for the libraries named in **-I** operands to look in  
89613        the directory named by the *directory* pathname before looking in the usual places.  
89614        Directories named in **-L** options shall be searched in the specified order. At least  
89615        ten instances of this option shall be supported in a single *fort77* command  
89616        invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the  
89617        results are unspecified.
- 89618        **-O** *optlevel*    Specify the level of code optimization. If the *optlevel* option-argument is the digit  
89619        '0', all special code optimizations shall be disabled. If it is the digit '1', the  
89620        nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
89621        the system's default optimization is unspecified. It is unspecified whether code  
89622        generated in the presence of the **-O 0** option is the same as that generated when  
89623        **-O** is omitted. Other *optlevel* values may be supported.
- 89624        **-w**                Suppress warnings.
- 89625        Multiple instances of **-L** options can be specified.

#### 89626 OPERANDS

- 89627        An *operand* is either in the form of a pathname or the form **-I library**. At least one operand of the  
89628        pathname form shall be specified. The following operands shall be supported:
- 89629        *file.f*            The pathname of a FORTRAN source file to be compiled and optionally passed to  
89630        the link editor. The filename operand shall be of this form if the **-c** option is used.
- 89631        *file.a*            A library of object files typically produced by *ar*, and passed directly to the link  
89632        editor. Implementations may recognize implementation-defined suffixes other  
89633        than **.a** as denoting object file libraries.
- 89634        *file.o*            An object file produced by *fort77 -c* and passed directly to the link editor.  
89635        Implementations may recognize implementation-defined suffixes other than **.o** as  
89636        denoting object files.
- 89637        The processing of other files is implementation-defined.
- 89638        **-I library**        (The letter ell.) Search the library named:  
89639        `liblibrary.a`
- 89640        A library is searched when its name is encountered, so the placement of a **-I**  
89641        operand is significant. Several standard libraries can be specified in this manner, as  
89642        described in the EXTENDED DESCRIPTION section. Implementations may  
89643        recognize implementation-defined suffixes other than **.a** as denoting libraries.

#### 89644 STDIN

- 89645        Not used.

#### 89646 INPUT FILES

- 89647        The input file shall be one of the following: a text file containing FORTRAN source code; an  
89648        object file in the format produced by *fort77 -c*; or a library of object files, in the format produced  
89649        by archiving zero or more object files, using *ar*. Implementations may supply additional utilities  
89650        that produce files in these formats. Additional input files are implementation-defined.
- 89651        A <tab> encountered within the first six characters on a line of source code shall cause the  
89652        compiler to interpret the following character as if it were the seventh character on the line (that  
89653        is, in column 7).

89654 **ENVIRONMENT VARIABLES**89655 The following environment variables shall affect the execution of *fort77*:

89656 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 89657 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 89658 variables used to determine the values of locale categories.)

89659 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 89660 internationalization variables.

89661 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 89662 characters (for example, single-byte as opposed to multi-byte characters in  
 89663 arguments and input files).

89664 *LC\_MESSAGES*

89665 Determine the locale that should be used to affect the format and contents of  
 89666 diagnostic messages written to standard error.

89667 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

89668 *TMPDIR* Determine the pathname that should override the default directory for temporary  
 89669 files, if any.

89670 **ASYNCHRONOUS EVENTS**

89671 Default.

89672 **STDOUT**

89673 Not used.

89674 **STDERR**

89675 The standard error shall be used only for diagnostic messages. If more than one *file* operand  
 89676 ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

89677 "%s:\n", <*file*>

89678 may be written to allow identification of the diagnostic message with the appropriate input file.

89679 This utility may produce warning messages about certain conditions that do not warrant  
 89680 returning an error (non-zero) exit value.

89681 **OUTPUT FILES**

89682 Object files, listing files, and executable files shall be produced in unspecified formats.

89683 **EXTENDED DESCRIPTION**89684 **Standard Libraries**89685 The *fort77* utility shall recognize the following *-l* operand for the standard library:

89686 *-l f* This library contains all functions referenced in the ANSI X3.9-1978 standard. This  
 89687 operand shall not be required to be present to cause a search of this library.

89688 In the absence of options that inhibit invocation of the link editor, such as *-c*, the *fort77* utility  
 89689 shall cause the equivalent of a *-l f* operand to be passed to the link editor as the last *-l* operand,  
 89690 causing it to be searched after all other object files and libraries are loaded.

89691 It is unspecified whether the library *libf.a* exists as a regular file. The implementation may  
 89692 accept as *-l* operands names of objects that do not exist as regular files.

89693 **External Symbols**

89694 The FORTRAN compiler and link editor shall support the significance of external symbols up to  
89695 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering  
89696 symbols exceeding the implementation-defined maximum symbol length is unspecified.

89697 The compiler and link editor shall support a minimum of 511 external symbols per source or  
89698 object file, and a minimum of 4095 external symbols total. A diagnostic message is written to  
89699 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

89700 **EXIT STATUS**

89701 The following exit values shall be returned:

89702 0 Successful compilation or link edit.

89703 >0 An error occurred.

89704 **CONSEQUENCES OF ERRORS**

89705 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and  
89706 continue to compile other source code operands. It shall return a non-zero exit status, but it is  
89707 implementation-defined whether an object module is created. If the link edit is unsuccessful, a  
89708 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero status.

89709 **APPLICATION USAGE**

89710 None.

89711 **EXAMPLES**

89712 The following usage example compiles *xyz.f* and creates the executable file *foo*:

89713 `fort77 -o foo xyz.f`

89714 The following example compiles *xyz.f* and creates the object file *xyz.o*:

89715 `fort77 -c xyz.f`

89716 The following example compiles *xyz.f* and creates the executable file *a.out*:

89717 `fort77 xyz.f`

89718 The following example compiles *xyz.f*, links it with *b.o*, and creates the executable *a.out*:

89719 `fort77 xyz.f b.o`

89720 **RATIONALE**

89721 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The  
89722 name *f77* was not chosen to avoid problems with historical implementations. The  
89723 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of  
89724 FORTRAN-77 has been superseded by the ISO/IEC 1539:1991 standard.

89725 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not  
89726 included in this volume of POSIX.1-2008—even though they are commonly implemented—since  
89727 there is no requirement that the FORTRAN compiler use the C preprocessor.

89728 The **-onetrip** option was not included in this volume of POSIX.1-2008, even though many  
89729 historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism  
89730 that should not be perpetuated.

89731 Some implementations produce compilation listings. This aspect of FORTRAN has been left  
89732 unspecified because there was controversy concerning the various methods proposed for  
89733 implementing it: a **-V** option overlapped with historical vendor practice and a naming  
89734 convention of creating files with *.l* suffixes collided with historical *lex* file naming practice.

- 89735 There is no **-I** option in this version of this volume of POSIX.1-2008 to specify a directory for file  
89736 inclusion. An **INCLUDE** directive has been a part of the Fortran-90 discussions, but an interface  
89737 supporting that standard is not in the current scope.
- 89738 It is noted that many FORTRAN compilers produce an object module even when compilation  
89739 errors occur; during a subsequent compilation, the compiler may patch the object module rather  
89740 than recompiling all the code. Consequently, it is left to the implementor whether or not an  
89741 object file is created.
- 89742 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from  
89743 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard  
89744 developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not  
89745 specify the military standard or any special preprocessing requirements. Furthermore, use of  
89746 that document would have been inappropriate for an international standard.
- 89747 The specification of optimization has been subject to changes through early proposals. At one  
89748 time, **-O** and **-N** were Booleans: optimize and do not optimize (with an unspecified default).  
89749 Some historical practice led this to be changed to:
- 89750 **-O 0** No optimization.
  - 89751 **-O 1** Some level of optimization.
  - 89752 **-O n** Other, unspecified levels of optimization.
- 89753 It is not always clear whether “good code generation” is the same thing as optimization. Simple  
89754 optimizations of local actions do not usually affect the semantics of a program. The **-O 0** option  
89755 has been included to accommodate the very particular nature of scientific calculations in a  
89756 highly optimized environment; compilers make errors. Some degree of optimization is expected,  
89757 even if it is not documented here, and the ability to shut it off completely could be important  
89758 when porting an application. An implementation may treat **-O 0** as “do less than normal” if it  
89759 wishes, but this is only meaningful if any of the operations it performs can affect the semantics  
89760 of a program. It is highly dependent on the implementation whether doing less than normal is  
89761 logical. It is not the intent of the **-O 0** option to ask for inefficient code generation, but rather to  
89762 assure that any semantically visible optimization is suppressed.
- 89763 The specification of standard library access is consistent with the C compiler specification.  
89764 Implementations are not required to have **/usr/lib/libf.a**, as many historical implementations do,  
89765 but if not they are required to recognize **f** as a token.
- 89766 External symbol size limits are in normative text; conforming applications need to know these  
89767 limits. However, the minimum maximum symbol length should be taken as a constraint on a  
89768 conforming application, not on an implementation, and consequently the action taken for a  
89769 symbol exceeding the limit is unspecified. The minimum size for the external symbol table was  
89770 added for similar reasons.
- 89771 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when  
89772 compilation or link-edit errors occur. The behavior of several historical implementations was  
89773 examined, and the choice was made to be silent on the status of the executable, or **a.out**, file in  
89774 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk  
89775 with *lseek()*s and *write()*s, the partially linked executable file can be left on disk and its execute  
89776 bits turned off if the link edit fails. However, if the linker links the image in memory before  
89777 writing the file to disk, it need not touch the executable file (if it already exists) because the link  
89778 edit fails. Since both approaches are historical practice, a conforming application shall rely on  
89779 the exit status of *fort77*, rather than on the existence or mode of the executable file.
- 89780 The **-g** and **-s** options are not specified as mutually-exclusive. Historically, these two options

- 89781 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate  
89782 to leave their interaction unspecified.
- 89783 The requirement that conforming applications specify compiler options separately is to reserve  
89784 the multi-character option name space for vendor-specific compiler options, which are known to  
89785 exist in many historical implementations. Implementations are not required to recognize, for  
89786 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all  
89787 of the options separately to highlight this requirement on applications.
- 89788 Echoing filenames to standard error is considered a diagnostic message because it would  
89789 otherwise be difficult to associate an error message with the erring file. They are described with  
89790 "may" to allow implementations to use other methods of identifying files and to parallel the  
89791 description in *c99*.
- 89792 **FUTURE DIRECTIONS**
- 89793 A compilation system based on the ISO/IEC 1539:1991 standard may be considered for a future  
89794 version; it may have a different utility name from *fort77*.
- 89795 **SEE ALSO**
- 89796 *ar, asa, c99, umask*
- 89797 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 89798 XSH *exec*
- 89799 **CHANGE HISTORY**
- 89800 First released in Issue 4.
- 89801 **Issue 6**
- 89802 This utility is marked as part of the FORTRAN Development Utilities option.
- 89803 The normative text is reworded to avoid use of the term "must" for application requirements.
- 89804 **Issue 7**
- 89805 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89806 **NAME**

89807 fuser — list process IDs of all processes that have one or more files open

89808 **SYNOPSIS**89809 XSI `fuser [-cfu] file...`89810 **DESCRIPTION**89811 The *fuser* utility shall write to standard output the process IDs of processes running on the local  
89812 system that have one or more named files open. For block special devices, all processes using  
89813 any file on that device are listed.89814 The *fuser* utility shall write to standard error additional information about the named files  
89815 indicating how the file is being used.

89816 Any output for processes running on remote systems that have a named file open is unspecified.

89817 A user may need appropriate privileges to invoke the *fuser* utility.89818 **OPTIONS**89819 The *fuser* utility shall conform to XBD Section 12.2 (on page 215).

89820 The following options shall be supported:

89821 **-c** The file is treated as a mount point and the utility shall report on any files open in  
89822 the file system.89823 **-f** The report shall be only for the named files.89824 **-u** The user name, in parentheses, associated with each process ID written to standard  
89825 output shall be written to standard error.89826 **OPERANDS**

89827 The following operand shall be supported:

89828 *file* A pathname on which the file or file system is to be reported.89829 **STDIN**

89830 Not used.

89831 **INPUT FILES**

89832 The user database.

89833 **ENVIRONMENT VARIABLES**89834 The following environment variables shall affect the execution of *fuser*:89835 **LANG** Provide a default value for the internationalization variables that are unset or null.  
89836 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
89837 variables used to determine the values of locale categories.)89838 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
89839 internationalization variables.89840 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
89841 characters (for example, single-byte as opposed to multi-byte characters in  
89842 arguments).89843 **LC\_MESSAGES**89844 Determine the locale that should be used to affect the format and contents of  
89845 diagnostic messages written to standard error.

|       |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 89846 | <i>NLSPATH</i>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                      |
| 89847 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89848 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 89849 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89850 |                               | The <i>fuser</i> utility shall write the process ID for each process using each file given as an operand to standard output in the following format:                                                                                                                                                                                                                                                                       |
| 89851 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89852 |                               | "%d", < <i>process_id</i> >                                                                                                                                                                                                                                                                                                                                                                                                |
| 89853 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89854 |                               | The <i>fuser</i> utility shall write diagnostic messages to standard error.                                                                                                                                                                                                                                                                                                                                                |
| 89855 |                               | The <i>fuser</i> utility also shall write the following to standard error:                                                                                                                                                                                                                                                                                                                                                 |
| 89856 |                               | <ul style="list-style-type: none"> <li>• The pathname of each named file is written followed immediately by a &lt;colon&gt;.</li> </ul>                                                                                                                                                                                                                                                                                    |
| 89857 |                               | <ul style="list-style-type: none"> <li>• For each process ID written to standard output, the character 'c' shall be written to standard error if the process is using the file as its current directory and the character 'r' shall be written to standard error if the process is using the file as its root directory. Implementations may write other alphabetic characters to indicate other uses of files.</li> </ul> |
| 89858 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89859 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89860 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89861 |                               | <ul style="list-style-type: none"> <li>• When the <i>-u</i> option is specified, characters indicating the use of the file shall be followed immediately by the user name, in parentheses, corresponding to the real user ID of the process. If the user name cannot be resolved from the real user ID of the process, the real user ID of the process shall be written instead of the user name.</li> </ul>               |
| 89862 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89863 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89864 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89865 |                               | When standard output and standard error are directed to the same file, the output shall be interleaved so that the filename appears at the start of each line, followed by the process ID and characters indicating the use of the file. Then, if the <i>-u</i> option is specified, the user name or user ID for each process using that file shall be written.                                                           |
| 89866 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89867 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89868 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89869 |                               | A <newline> shall be written to standard error after the last output described above for each <i>file</i> operand.                                                                                                                                                                                                                                                                                                         |
| 89870 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89871 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89872 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 89873 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89874 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 89875 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89876 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                               |
| 89877 |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                                   |
| 89878 |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                                      |
| 89879 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 89880 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                   |

89881 **APPLICATION USAGE**

89882 None.

89883 **EXAMPLES**

89884 The command:

89885 `fuser -fu .`

89886 writes to standard output the process IDs of processes that are using the current directory and  
 89887 writes to standard error an indication of how those processes are using the directory and the  
 89888 user names associated with the processes that are using the current directory.

89889 `fuser -c <mount point>`

89890 writes to standard output the process IDs of processes that are using any file in the file system  
 89891 which is mounted on <mount point> and writes to standard error an indication of how those  
 89892 processes are using the files.

89893 `fuser <mount point>`

89894 writes to standard output the process IDs of processes that are using the file which is named by  
 89895 <mount point> and writes to standard error an indication of how those processes are using the  
 89896 file.

89897 `fuser <block device>`

89898 writes to standard output the process IDs of processes that are using any file which is on the  
 89899 device named by <block device> and writes to standard error an indication of how those  
 89900 processes are using the file.

89901 `fuser --f <block device>`

89902 writes to standard output the process IDs of processes that are using the file <block device> itself  
 89903 and writes to standard error an indication of how those processes are using the file.

89904 **RATIONALE**89905 The definition of the *fuser* utility follows existing practice.89906 **FUTURE DIRECTIONS**

89907 None.

89908 **SEE ALSO**89909 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)89910 **CHANGE HISTORY**

89911 First released in Issue 5.

89912 **Issue 7**

89913 SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

89914 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**gencat**

Utilities

89915 **NAME**89916 `gencat` — generate a formatted message catalog89917 **SYNOPSIS**89918 `gencat catfile msgfile...`89919 **DESCRIPTION**

89920 The `gencat` utility shall merge the message text source file `msgfile` into a formatted message  
 89921 catalog `catfile`. The file `catfile` shall be created if it does not already exist. If `catfile` does exist, its  
 89922 messages shall be included in the new `catfile`. If set and message numbers collide, the new  
 89923 message text defined in `msgfile` shall replace the old message text currently contained in `catfile`.

89924 **OPTIONS**

89925 None.

89926 **OPERANDS**

89927 The following operands shall be supported:

89928 `catfile` A pathname of the formatted message catalog. If `'-'` is specified, standard output  
 89929 shall be used. The format of the message catalog produced is unspecified.

89930 `msgfile` A pathname of a message text source file. If `'-'` is specified for an instance of  
 89931 `msgfile`, standard input shall be used. The format of message text source files is  
 89932 defined in the EXTENDED DESCRIPTION section.

89933 **STDIN**89934 The standard input shall not be used unless a `msgfile` operand is specified as `'-'`.89935 **INPUT FILES**

89936 The input files shall be text files.

89937 **ENVIRONMENT VARIABLES**89938 The following environment variables shall affect the execution of `gencat`:

89939 `LANG` Provide a default value for the internationalization variables that are unset or null.  
 89940 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 89941 variables used to determine the values of locale categories.)

89942 `LC_ALL` If set to a non-empty string value, override the values of all the other  
 89943 internationalization variables.

89944 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
 89945 characters (for example, single-byte as opposed to multi-byte characters in  
 89946 arguments and input files).

89947 `LC_MESSAGES`

89948 Determine the locale that should be used to affect the format and contents of  
 89949 diagnostic messages written to standard error.

89950 `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

89951 **ASYNCHRONOUS EVENTS**

89952 Default.

89953 **STDOUT**89954 The standard output shall not be used unless the `catfile` operand is specified as `'-'`.

89955 **STDERR**

89956 The standard error shall be used only for diagnostic messages.

89957 **OUTPUT FILES**

89958 None.

89959 **EXTENDED DESCRIPTION**89960 The content of a message text file shall be in the format defined as follows. Note that the fields of  
89961 a message text source line are separated by a single <blank> character. Any other <blank>  
89962 characters are considered to be part of the subsequent field.89963 **\$set** *n comment*89964 This line specifies the set identifier of the following messages until the next **\$set** or  
89965 end-of-file appears. The *n* denotes the set identifier, which is defined as a number  
89966 in the range [1, {NL\_SETMAX}] (see the <limits.h> header defined in the Base  
89967 Definitions volume of POSIX.1-2008). The application shall ensure that set  
89968 identifiers are presented in ascending order within a single source file, but need  
89969 not be contiguous. Any string following the set identifier shall be treated as a  
89970 comment. If no **\$set** directive is specified in a message text source file, all messages  
89971 shall be located in an implementation-defined default message set NL\_SETD (see  
89972 the <nl\_types.h> header defined in the Base Definitions volume of POSIX.1-2008).89973 **\$delset** *n comment*89974 This line deletes message set *n* from an existing message catalog. The *n* denotes the  
89975 set number [1, {NL\_SETMAX}]. Any string following the set number shall be  
89976 treated as a comment.89977 **\$ comment** A line beginning with ' \$ ' followed by a <blank> shall be treated as a comment.89978 *m message-text*89979 The *m* denotes the message identifier, which is defined as a number in the range [1,  
89980 {NL\_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the  
89981 message catalog with the set identifier specified by the last **\$set** directive, and with  
89982 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is  
89983 present, an empty string shall be stored in the message catalog. If a message source  
89984 line has a message number, but neither a field separator nor *message-text*, the  
89985 existing message with that number (if any) shall be deleted from the catalog. The  
89986 application shall ensure that message identifiers are in ascending order within a  
89987 single set, but need not be contiguous. The application shall ensure that the length  
89988 of *message-text* is in the range [0, {NL\_TEXTMAX}] (see the <limits.h> header).89989 **\$quote** *n* This line specifies an optional quote character *c*, which can be used to surround  
89990 *message-text* so that trailing <space> characters or null (empty) messages are visible  
89991 in a message source line. By default, or if an empty **\$quote** directive is supplied, no  
89992 quoting of *message-text* shall be recognized.89993 Empty lines in a message text source file shall be ignored. The effects of lines starting with any  
89994 character other than those defined above are implementation-defined.89995 Text strings can contain the special characters and escape sequences defined in the following  
89996 table:

**gencat**

89997  
89998  
89999  
90000  
90001  
90002  
90003  
90004  
90005

| Description       | Symbol | Sequence |
|-------------------|--------|----------|
| <newline>         | NL(LF) | \n       |
| Horizontal-tab    | HT     | \t       |
| <vertical-tab>    | VT     | \v       |
| <backspace>       | BS     | \b       |
| <carriage-return> | CR     | \r       |
| <form-feed>       | FF     | \f       |
| Backslash         | \      | \\       |
| Bit pattern       | ddd    | \ddd     |

90006 The escape sequence "\ddd" consists of <backslash> followed by one, two, or three octal digits,  
90007 which shall be taken to specify the value of the desired character. If the character following a  
90008 <backslash> is not one of those specified, the <backslash> shall be ignored.

90009 A <backslash> followed by a <newline> is also used to continue a string on the following line.  
90010 Thus, the following two lines describe a single message string:

90011 1 This line continues \  
90012 to the next line

90013 which shall be equivalent to:

90014 1 This line continues to the next line

90015 **EXIT STATUS**

90016 The following exit values shall be returned:

90017 0 Successful completion.

90018 >0 An error occurred.

90019 **CONSEQUENCES OF ERRORS**

90020 Default.

90021 **APPLICATION USAGE**

90022 Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot  
90023 be guaranteed between different types of machine. Thus, just as C programs need to be  
90024 recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

90025 **EXAMPLES**

90026 None.

90027 **RATIONALE**

90028 None.

90029 **FUTURE DIRECTIONS**

90030 None.

90031 **SEE ALSO**

90032 *iconv*

90033 XBD Chapter 8 (on page 173), <limits.h>, <nl\_types.h>

90034 **CHANGE HISTORY**

90035 First released in Issue 3.

90036 **Issue 6**

90037 The normative text is reworded to avoid use of the term “must” for application requirements.

90038 **Issue 7**  
90039

The *gencat* utility is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**get**90040 **NAME**90041 `get` — get a version of an SCCS file (**DEVELOPMENT**)90042 **SYNOPSIS**90043 xSI `get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`90044 **DESCRIPTION**90045 The `get` utility shall generate a text file from each named SCCS *file* according to the specifications  
90046 given by its options.90047 The generated text shall normally be written into a file called the **g-file** whose name is derived  
90048 from the SCCS filename by simply removing the leading "s.". 90049 **OPTIONS**90050 The `get` utility shall conform to XBD Section 12.2 (on page 215).

90051 The following options shall be supported:

90052 **-r *SID*** Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file  
90053 to be retrieved. The table shows, for the most useful cases, what version of an  
90054 SCCS file is retrieved (as well as the SID of the version to be eventually created by  
90055 *delta* if the **-e** option is also used), as a function of the SID specified.90056 **-c *cutoff*** Indicate the *cutoff* date-time, in the form:90057 `YY[MM[DD[HH[MM[SS]]]]]`90058 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999  
90059 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.90060 **Note:** It is expected that in a future version of this standard the default century inferred  
90061 from a 2-digit year will change. (This would apply to all commands accepting a  
90062 2-digit year as input.)90063 No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
90064 date-time shall be included in the generated text file. Units omitted from the date-  
90065 time default to their maximum possible values; for example, **-c 7502** is equivalent  
90066 to **-c 750228235959**.90067 Any number of non-numeric characters may separate the various 2-digit pieces of  
90068 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:  
90069 **-c "77/2/2 9:22:25"**.90070 **-e** Indicate that the `get` is for the purpose of editing or making a change (delta) to the  
90071 SCCS file via a subsequent use of *delta*. The **-e** option used in a `get` for a particular  
90072 version (SID) of the SCCS file shall prevent further `get` commands from editing on  
90073 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.  
90074 Concurrent use of `get -e` for different SIDs is always allowed.90075 If the **g-file** generated by `get` with a **-e** option is accidentally ruined in the process  
90076 of editing, it may be regenerated by re-executing the `get` command with the **-k**  
90077 option in place of the **-e** option.90078 SCCS file protection specified via the ceiling, floor, and authorized user list stored  
90079 in the SCCS file shall be enforced when the **-e** option is used.90080 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new  
90081 branch as shown in the table below. This option shall be ignored if the **b** flag is not  
90082 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that  
90083 has no successors on the SCCS file tree.)

- 90084                   **Note:**     A branch delta may always be created from a non-leaf delta.
- 90085           **-i list**     Indicate a *list* of deltas to be included (forced to be applied) in the creation of the  
90086                   generated file. The *list* has the following syntax:
- 90087                   <list> ::= <range> | <list> , <range>  
90088                   <range> ::= SID | SID - SID
- 90089                   SID, the SCCS Identification of a delta, may be in any form shown in the "SID  
90090                   Specified" column of the table in the EXTENDED DESCRIPTION section, except  
90091                   that the result of supplying a partial SID is unspecified. A diagnostic message shall  
90092                   be written if the first SID in the range is not an ancestor of the second SID in the  
90093                   range.
- 90094           **-x list**     Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of  
90095                   the generated file. See the **-i** option for the *list* format.
- 90096           **-k**         Suppress replacement of identification keywords (see below) in the retrieved text  
90097                   by their value. The **-k** option shall be implied by the **-e** option.
- 90098           **-l**         Write a delta summary into an **l-file**.
- 90099           **-L**         Write a delta summary to standard output. All informative output that normally is  
90100                   written to standard output shall be written to standard error instead, unless the **-s**  
90101                   option is used, in which case it shall be suppressed.
- 90102           **-p**         Write the text retrieved from the SCCS file to the standard output. No **g-file** shall  
90103                   be created. All informative output that normally goes to the standard output shall  
90104                   go to standard error instead, unless the **-s** option is used, in which case it shall  
90105                   disappear.
- 90106           **-s**         Suppress all informative output normally written to standard output. However,  
90107                   fatal error messages (which shall always be written to the standard error) shall  
90108                   remain unaffected.
- 90109           **-m**         Precede each text line retrieved from the SCCS file by the SID of the delta that  
90110                   inserted the text line in the SCCS file. The format shall be:
- 90111                   "%s\t%s", <SID>, <text line>
- 90112           **-n**         Precede each generated text line with the **%M%** identification keyword value (see  
90113                   below). The format shall be:
- 90114                   "%s\t%s", <%M% value>, <text line>
- 90115                   When both the **-m** and **-n** options are used, the <text line> shall be replaced by the  
90116                   **-m** option-generated format.
- 90117           **-g**         Suppress the actual retrieval of text from the SCCS file. It is primarily used to  
90118                   generate an **l-file**, or to verify the existence of a particular SID.
- 90119           **-t**         Use to access the most recently created (top) delta in a given release (for example,  
90120                   **-r 1**), or release and level (for example, **-r 1.2**).

90121 **OPERANDS**

90122 The following operands shall be supported:

90123 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*  
 90124 utility shall behave as though each file in the directory were specified as a named  
 90125 file, except that non-SCCS files (last component of the pathname does not begin  
 90126 with *s*.) and unreadable files shall be silently ignored.

90127 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;  
 90128 each line of the standard input is taken to be the name of an SCCS file to be  
 90129 processed. Non-SCCS files and unreadable files shall be silently ignored.

90130 **STDIN**

90131 The standard input shall be a text file used only if the *file* operand is specified as *'-'*. Each line  
 90132 of the text file shall be interpreted as an SCCS pathname.

90133 **INPUT FILES**

90134 The SCCS files shall be files of an unspecified format.

90135 **ENVIRONMENT VARIABLES**90136 The following environment variables shall affect the execution of *get*:

90137 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 90138 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 90139 variables used to determine the values of locale categories.)

90140 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 90141 internationalization variables.

90142 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 90143 characters (for example, single-byte as opposed to multi-byte characters in  
 90144 arguments and input files).

90145 *LC\_MESSAGES*

90146 Determine the locale that should be used to affect the format and contents of  
 90147 diagnostic messages written to standard error, and informative messages written  
 90148 to standard output (or standard error, if the *-p* option is used).

90149 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

90150 *TZ* Determine the timezone in which the times and dates written in the SCCS file are  
 90151 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default  
 90152 timezone is used.

90153 **ASYNCHRONOUS EVENTS**

90154 Default.

90155 **STDOUT**

90156 For each file processed, *get* shall write to standard output the SID being accessed and the  
 90157 number of lines retrieved from the SCCS file, in the following format:

90158 "%s\n%d lines\n", &lt;SID&gt;, &lt;number of lines&gt;

90159 If the *-e* option is used, the SID of the delta to be made shall appear after the SID accessed and  
 90160 before the number of lines generated, in the POSIX locale:

90161 "%s\nnew delta %s\n%d lines\n", <SID accessed>,  
 90162 <SID to be made>, <number of lines>

90163 If there is more than one named file or if a directory or standard input is named, each pathname

90164 shall be written before each of the lines shown in one of the preceding formats:

90165 `"\n%s:\n", <pathname>`

90166 If the `-L` option is used, a delta summary shall be written following the format specified below  
90167 for **l-files**.

90168 If the `-i` option is used, included deltas shall be listed following the notation, in the POSIX  
90169 locale:

90170 `"Included:\n"`

90171 If the `-x` option is used, excluded deltas shall be listed following the notation, in the POSIX  
90172 locale:

90173 `"Excluded:\n"`

90174 If the `-p` or `-L` options are specified, the standard output shall consist of the text retrieved from  
90175 the SCCS file.

#### 90176 **STDERR**

90177 The standard error shall be used only for diagnostic messages, except if the `-p` or `-L` options are  
90178 specified, it shall include all informative messages normally sent to standard output.

#### 90179 **OUTPUT FILES**

90180 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**  
90181 **file**, **p-file**, and **z-file**. The letter before the <hyphen> is called the *tag*. An auxiliary filename  
90182 shall be formed from the SCCS filename: the application shall ensure that the last component of  
90183 all SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named by replacing  
90184 the leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by  
90185 removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be *xyz.c*, **l.xyz.c**,  
90186 **p.xyz.c**, and **z.xyz.c**, respectively.

90187 The **g-file**, which contains the generated text, shall be created in the current directory (unless the  
90188 `-p` option is used). A **g-file** shall be created in all cases, whether or not any lines of text were  
90189 generated by the *get*. It shall be owned by the real user. If the `-k` option is used or implied, the  
90190 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be  
90191 read-only. Only the real user need have write permission in the current directory.

90192 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved  
90193 text. The **l-file** shall be created in the current directory if the `-l` option is used; it shall be read-  
90194 only and it is owned by the real user. Only the real user need have write permission in the  
90195 current directory.

90196 Lines in the **l-file** shall have the following format:

90197 `"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>, <SID>, <date-time>, <login>`

90198 where the entries are:

90200 `<code1>` A <space> if the delta was applied; ' \* ' otherwise.

90201 `<code2>` A <space> if the delta was applied or was not applied and ignored; ' \* ' if the delta  
90202 was not applied and was not ignored.

90203 `<code3>` A character indicating a special reason why the delta was or was not applied:

90204 **I** Included.

## get

## Utilities

- 90205                   X   Excluded.
- 90206                   C   Cut off (by a `-c` option).
- 90207           <*date-time*>   Date and time (using the format of the *date* utility's `%Y/%m/%d %T` conversion specification format) of creation.
- 90208
- 90209           <*login*>        Login name of person who created *delta*.
- 90210           The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line shall terminate each entry.
- 90211
- 90212           The **p-file** shall be used to pass information resulting from a *get* with a `-e` option along to *delta*. Its contents shall also be used to prevent a subsequent execution of *get* with a `-e` option for the same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be created in the directory containing the SCCS file and the application shall ensure that the effective user has write permission in that directory. It shall be writable by owner only, and owned by the effective user. Each line in the **p-file** shall have the following format:
- 90213
- 90214
- 90215
- 90216
- 90217
- 90218           "`%sΔ%sΔ%sΔ%s%s\n`", <*g-file SID*>,  
 90219                    <*SID of new delta*>, <*login-name of real user*>,  
 90220                    <*date-time*>, <*i-value*>, <*x-value*>
- 90221           where <*i-value*> uses the format " " if no `-i` option was specified, and shall use the format:
- 90222           "`Δ-i%s`", <*-i option option-argument*>
- 90223           if a `-i` option was specified and <*x-value*> uses the format " " if no `-x` option was specified, and shall use the format:
- 90224
- 90225           "`Δ-x%s`", <*-x option option-argument*>
- 90226           if a `-x` option was specified. There can be an arbitrary number of lines in the **p-file** at any time; no two lines shall have the same new delta SID.
- 90227
- 90228           The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.
- 90229
- 90230
- 90231

## 90232 EXTENDED DESCRIPTION

| Determination of SCCS Identification String |                    |                                             |               |                            |
|---------------------------------------------|--------------------|---------------------------------------------|---------------|----------------------------|
| SID* Specified                              | -b Keyletter Used† | Other Conditions                            | SID Retrieved | SID of Delta to be Created |
| 90233<br>90234<br>90235<br>none‡            | no                 | R defaults to mR                            | mR.mL         | mR.(mL+1)                  |
| 90236<br>90237<br>none‡                     | yes                | R defaults to mR                            | mR.mL         | mR.mL.(mB+1).1             |
| 90238<br>90239<br>R                         | no                 | R > mR                                      | mR.mL         | R.1***                     |
| 90240<br>90241<br>R                         | no                 | R = mR                                      | mR.mL         | mR.(mL+1)                  |
| 90242<br>90243<br>R                         | yes                | R > mR                                      | mR.mL         | mR.mL.(mB+1).1             |
| 90244<br>90245<br>R                         | yes                | R = mR                                      | mR.mL         | mR.mL.(mB+1).1             |
| 90246<br>90247<br>R                         | –                  | R < mR and R does not exist                 | hR.mL**       | hR.mL.(mB+1).1             |
| 90248<br>90249<br>R                         | –                  | Trunk successor in release > R and R exists | R.mL          | R.mL.(mB+1).1              |
| 90250<br>90251<br>R.L                       | no                 | No trunk successor                          | R.L           | R.(L+1)                    |
| 90252<br>90253<br>R.L                       | yes                | No trunk successor                          | R.L           | R.L.(mB+1).1               |
| 90254<br>90255<br>R.L                       | –                  | Trunk successor in release ≥ R              | R.L           | R.L.(mB+1).1               |
| 90256<br>90257<br>R.L.B                     | no                 | No branch successor                         | R.L.B.mS      | R.L.B.(mS+1)               |
| 90258<br>90259<br>R.L.B                     | yes                | No branch successor                         | R.L.B.mS      | R.L.(mB+1).1               |
| 90260<br>90261<br>R.L.B.S                   | no                 | No branch successor                         | R.L.B.S       | R.L.B.(S+1)                |
| 90262<br>90263<br>R.L.B.S                   | yes                | No branch successor                         | R.L.B.S       | R.L.(mB+1).1               |
| 90264<br>90265<br>R.L.B.S                   | –                  | Branch successor                            | R.L.B.S       | R.L.(mB+1).1               |

90255 \* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means “the maximum level number within release R”; R.L.(mB+1).1 means “the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R”. Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

90261 \*\* hR is the highest existing release that is lower than the specified, nonexistent, release R.

90262 \*\*\* This is used to force creation of the first delta in a new release.

90263 † The -b option is effective only if the b flag is present in the file. An entry of ‘-’ means “irrelevant”.

90265 ‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

90268 **System Date and Time**

90269 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into  
 90270 account. If any of these times are apparently in the future, the behavior is unspecified.

90271 **Identification Keywords**

90272 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing  
 90273 identification keywords with their value wherever they occur. The following keywords may be  
 90274 used in the text stored in an SCCS file:

|       |     |                                                                                               |
|-------|-----|-----------------------------------------------------------------------------------------------|
| 90275 | %M% | Module name: either the value of the <b>m</b> flag in the file, or if absent, the name of the |
| 90276 |     | SCCS file with the leading <b>s.</b> removed.                                                 |
| 90277 | %I% | SCCS identification (SID) (%R%.%L% or %R%.%L%.%B%.%S%) of the retrieved                       |
| 90278 |     | text.                                                                                         |
| 90279 | %R% | Release.                                                                                      |
| 90280 | %L% | Level.                                                                                        |
| 90281 | %B% | Branch.                                                                                       |
| 90282 | %S% | Sequence.                                                                                     |
| 90283 | %D% | Current date (YY/MM/DD).                                                                      |
| 90284 | %H% | Current date (MM/DD/YY).                                                                      |
| 90285 | %T% | Current time (HH:MM:SS).                                                                      |
| 90286 | %E% | Date newest applied delta was created (YY/MM/DD).                                             |
| 90287 | %G% | Date newest applied delta was created (MM/DD/YY).                                             |
| 90288 | %U% | Time newest applied delta was created (HH:MM:SS).                                             |
| 90289 | %Y% | Module type: value of the <b>t</b> flag in the SCCS file.                                     |
| 90290 | %F% | SCCS filename.                                                                                |
| 90291 | %P% | SCCS absolute pathname.                                                                       |
| 90292 | %Q% | The value of the <b>q</b> flag in the file.                                                   |
| 90293 | %C% | Current line number. This keyword is intended for identifying messages output by              |
| 90294 |     | the program, such as "this should not have happened" type errors. It is not                   |
| 90295 |     | intended to be used on every line to provide sequence numbers.                                |
| 90296 | %Z% | The four-character string "@ (#)" recognizable by <i>what</i> .                               |
| 90297 | %W% | A shorthand notation for constructing <i>what</i> strings:                                    |
| 90298 |     | %W%=%Z%%M%<tab>%I%                                                                            |
| 90299 | %A% | Another shorthand notation for constructing <i>what</i> strings:                              |
| 90300 |     | %A%=%Z%%Y%%M%%I%%Z%                                                                           |

90301 **EXIT STATUS**

90302 The following exit values shall be returned:

90303 0 Successful completion.

90304 >0 An error occurred.

90305 **CONSEQUENCES OF ERRORS**

90306 Default.

90307 **APPLICATION USAGE**

90308 Problems can arise if the system date and time have been modified (for example, put forward  
90309 and then back again, or unsynchronized clocks across a network) and can also arise when  
90310 different values of the *TZ* environment variable are used.

90311 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares  
90312 the previous file body against the working file as part of its normal operation.

90313 **EXAMPLES**

90314 None.

90315 **RATIONALE**

90316 None.

90317 **FUTURE DIRECTIONS**

90318 None.

90319 **SEE ALSO**

90320 *admin, delta, prs, what*

90321 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

90322 **CHANGE HISTORY**

90323 First released in Issue 2.

90324 **Issue 5**

90325 A correction is made to the first format string in STDOUT.

90326 The interpretation of the *YY* component of the *-c cutoff* argument is noted.

90327 **Issue 6**

90328 The obsolescent SYNOPSIS is removed, removing the *-lp* option.

90329 The normative text is reworded to avoid use of the term “must” for application requirements.

90330 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

90331 The Open Group Corrigendum U048/1 is applied.

90332 The Open Group Interpretation PIN4C.00014 is applied.

90333 The Open Group Base Resolution bwg2001-007 is applied as follows:

90334 • The EXTENDED DESCRIPTION section is updated to make partial SID handling  
90335 unspecified, reflecting common usage, and to clarify SID ranges.

90336 • New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections  
90337 regarding how the system date and time may be taken into account.

90338 • The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

90339 **Issue 7**

90340 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**getconf**

Utilities

90341 **NAME**90342 `getconf` — get configuration values90343 **SYNOPSIS**90344 `getconf [-v specification] system_var`90345 `getconf [-v specification] path_var pathname`90346 **DESCRIPTION**90347 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the  
90348 variable specified by the *system\_var* operand.90349 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the  
90350 variable specified by the *path\_var* operand for the path specified by the *pathname* operand.90351 The value of each configuration variable shall be determined as if it were obtained by calling the  
90352 function from which it is defined to be available by this volume of POSIX.1-2008 or by the  
90353 System Interfaces volume of POSIX.1-2008 (see the OPERANDS section). The value shall reflect  
90354 conditions in the current operating environment.90355 **OPTIONS**90356 The *getconf* utility shall conform to XBD Section 12.2 (on page 215).

90357 The following option shall be supported:

90358 **-v specification**90359 Indicate a specific specification and version for which configuration variables shall  
90360 be determined. If this option is not specified, the values returned correspond to an  
90361 implementation default conforming compilation environment.

90362 If the command:

90363 `getconf _POSIX_V7_ILP32_OFF32`90364 does not write "-1\n" or "undefined\n" to standard output, then commands of  
90365 the form:90366 `getconf -v POSIX_V7_ILP32_OFF32 ...`90367 determine values for configuration variables corresponding to the  
90368 POSIX\_V7\_ILP32\_OFF32 compilation environment specified in c99, the  
90369 EXTENDED DESCRIPTION.

90370 If the command:

90371 `getconf _POSIX_V7_ILP32_OFFBIG`90372 does not write "-1\n" or "undefined\n" to standard output, then commands of  
90373 the form:90374 `getconf -v POSIX_V7_ILP32_OFFBIG ...`90375 determine values for configuration variables corresponding to the  
90376 POSIX\_V7\_ILP32\_OFFBIG compilation environment specified in c99, the  
90377 EXTENDED DESCRIPTION.

90378 If the command:

90379 `getconf _POSIX_V7_LP64_OFF64`90380 does not write "-1\n" or "undefined\n" to standard output, then commands of  
90381 the form:

90382 `getconf -v POSIX_V7_LP64_OFF64 ...`

90383 determine values for configuration variables corresponding to the  
90384 POSIX\_V7\_LP64\_OFF64 compilation environment specified in *c99*, the  
90385 EXTENDED DESCRIPTION.

90386 If the command:

90387 `getconf _POSIX_V7_LPBIG_OFFBIG`

90388 does not write "-1\n" or "undefined\n" to standard output, then commands of  
90389 the form:

90390 `getconf -v POSIX_V7_LPBIG_OFFBIG ...`

90391 determine values for configuration variables corresponding to the  
90392 POSIX\_V7\_LPBIG\_OFFBIG compilation environment specified in *c99*, the  
90393 EXTENDED DESCRIPTION.

#### 90394 OPERANDS

90395 The following operands shall be supported:

90396 *path\_var* A name of a configuration variable. All of the variables in the Variable column of  
90397 the table in the DESCRIPTION of the *fpathconf()* function defined in the System  
90398 Interfaces volume of POSIX.1-2008, without the enclosing braces, shall be  
90399 supported. The implementation may add other local variables.

90400 *pathname* A pathname for which the variable specified by *path\_var* is to be determined.

90401 *system\_var* A name of a configuration variable. All of the following variables shall be  
90402 supported:

- 90403 • The names in the Variable column of the table in the DESCRIPTION of the
- 90404 *sysconf()* function in the System Interfaces volume of POSIX.1-2008, except
- 90405 for the entries corresponding to `_SC_CLK_TCK`, `_SC_GETGR_R_SIZE_MAX`,
- 90406 and `_SC_GETPW_R_SIZE_MAX`, without the enclosing braces.

90407 For compatibility with earlier versions, the following variable names shall  
90408 also be supported:

90409 `POSIX2_C_BIND`  
90410 `POSIX2_C_DEV`  
90411 `POSIX2_CHAR_TERM`  
90412 `POSIX2_FORT_DEV`  
90413 `POSIX2_FORT_RUN`  
90414 `POSIX2_LOCALEDEF`  
90415 `POSIX2_SW_DEV`  
90416 `POSIX2_UPE`  
90417 `POSIX2_VERSION`

90418 and shall be equivalent to the same name prefixed with an <underscore>.  
90419 This requirement may be removed in a future version.

- 90420 • The names of the symbolic constants used as the *name* argument of the
- 90421 *confstr()* function in the System Interfaces volume of POSIX.1-2008, without
- 90422 the `_CS_` prefix.

**getconf**

Utilities

- 90423 • The names of the symbolic constants listed under the headings “Maximum  
90424 Values” and “Minimum Values” in the description of the `<limits.h>` header  
90425 in the Base Definitions volume of POSIX.1-2008, without the enclosing  
90426 braces.
- 90427 For compatibility with earlier versions, the following variable names shall  
90428 also be supported:
- 90429 POSIX2\_BC\_BASE\_MAX  
90430 POSIX2\_BC\_DIM\_MAX  
90431 POSIX2\_BC\_SCALE\_MAX  
90432 POSIX2\_BC\_STRING\_MAX  
90433 POSIX2\_COLL\_WEIGHTS\_MAX  
90434 POSIX2\_EXPR\_NEST\_MAX  
90435 POSIX2\_LINE\_MAX  
90436 POSIX2\_RE\_DUP\_MAX
- 90437 and shall be equivalent to the same name prefixed with an `<underscore>`.  
90438 This requirement may be removed in a future version.
- 90439 The implementation may add other local values.
- 90440 **STDIN**  
90441 Not used.
- 90442 **INPUT FILES**  
90443 None.
- 90444 **ENVIRONMENT VARIABLES**  
90445 The following environment variables shall affect the execution of `getconf`:
- 90446 `LANG` Provide a default value for the internationalization variables that are unset or null.  
90447 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
90448 variables used to determine the values of locale categories.)
- 90449 `LC_ALL` If set to a non-empty string value, override the values of all the other  
90450 internationalization variables.
- 90451 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
90452 characters (for example, single-byte as opposed to multi-byte characters in  
90453 arguments).
- 90454 `LC_MESSAGES`  
90455 Determine the locale that should be used to affect the format and contents of  
90456 diagnostic messages written to standard error.
- 90457 `XSI` `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.
- 90458 **ASYNCHRONOUS EVENTS**  
90459 Default.
- 90460 **STDOUT**  
90461 If the specified variable is defined on the system and its value is described to be available from  
90462 the `confstr()` function defined in the System Interfaces volume of POSIX.1-2008, its value shall be  
90463 written in the following format:  
90464 `"%s\n", <value>`  
90465 Otherwise, if the specified variable is defined on the system, its value shall be written in the

90466 following format:

90467 "%d\n", <value>

90468 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the  
90469 following format:

90470 "undefined\n"

90471 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

#### 90472 **STDERR**

90473 The standard error shall be used only for diagnostic messages.

#### 90474 **OUTPUT FILES**

90475 None.

#### 90476 **EXTENDED DESCRIPTION**

90477 None.

#### 90478 **EXIT STATUS**

90479 The following exit values shall be returned:

90480 0 The specified variable is valid and information about its current state was written  
90481 successfully.

90482 >0 An error occurred.

#### 90483 **CONSEQUENCES OF ERRORS**

90484 Default.

#### 90485 **APPLICATION USAGE**

90486 None.

#### 90487 **EXAMPLES**

90488 The following example illustrates the value of {NGROUPS\_MAX}:

90489 `getconf NGROUPS_MAX`

90490 The following example illustrates the value of {NAME\_MAX} for a specific directory:

90491 `getconf NAME_MAX /usr`

90492 The following example shows how to deal more carefully with results that might be unspecified:

```
90493 if value=$(getconf PATH_MAX /usr); then
90494     if [ "$value" = "undefined" ]; then
90495         echo PATH_MAX in /usr is indeterminate.
90496     else
90497         echo PATH_MAX in /usr is $value.
90498     fi
90499 else
90500     echo Error in getconf.
90501 fi
```

90502 Note that:

90503 `sysconf (_SC_2_C_BIND) ;`

90504 and:

90505 `system("getconf _POSIX2_C_BIND") ;`

**getconf**

90506 in a C program could give different answers. The *sysconf()* call supplies a value that corresponds  
 90507 to the conditions when the program was either compiled or executed, depending on the  
 90508 implementation; the *system()* call to *getconf* always supplies a value corresponding to conditions  
 90509 when the program is executed.

**RATIONALE**

90510 The original need for this utility, and for the *confstr()* function, was to provide a way of finding  
 90511 the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be  
 90512 modified by the user to include directories that could contain utilities replacing the standard  
 90513 utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable  
 90514 value that contains the correct search path for the standard utilities. It was later suggested that  
 90515 access to the other variables described in this volume of POSIX.1-2008 could also be useful to  
 90516 applications.  
 90517

90518 This functionality of *getconf* would not be adequately subsumed by another command such as:

```
90519 grep var /etc/conf
```

90520 because such a strategy would provide correct values for neither those variables that can vary at  
 90521 runtime, nor those that can vary depending on the path.

90522 Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid,  
 90523 but not defined on the system. The output string "undefined" is now used to specify this case  
 90524 with exit code 0 because so many things depend on an exit code of zero when an invoked utility  
 90525 is successful.

**FUTURE DIRECTIONS**

90526 None.  
 90527

**SEE ALSO**

90528 *c99*

90530 XBD Chapter 8 (on page 173), Section 12.2 (on page 215), <limits.h>

90531 XSH *confstr()*, *fpathconf()*, *sysconf()*, *system()*

**CHANGE HISTORY**

90532 First released in Issue 4

**Issue 5**

90533 In the OPERANDS section:

- 90536 • {NL\_MAX} is changed to {NL\_NMAX}.
- 90537 • Entries beginning NL\_ are deleted from the list of standard configuration variables.
- 90538 • The list of variables previously marked UX is merged with the list marked EX.
- 90539 • Operands are added to support new Option Groups.
- 90540 • Operands are added so that *getconf* can determine supported programming environments.

**Issue 6**

90541 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last  
 90542 paragraph of the OPTIONS section.  
 90543

90544 The following new requirements on POSIX implementations derive from alignment with the  
 90545 Single UNIX Specification:

- 90546           • Operands are added to determine supported programming environments.
- 90547           This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically,  
90548           new macros for *c99* programming environments are introduced.
- 90549           XSI marked *system\_var* (XBS5\_\*) values are marked LEGACY.
- 90550           IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions  
90551           of *path\_var* and *system\_var* in the OPERANDS section.
- 90552   **Issue 7**
- 90553           SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 90554           The EXAMPLES section is corrected.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**getopts**

Utilities

90555 **NAME**90556 `getopts` — parse utility options90557 **SYNOPSIS**90558 `getopts optstring name [arg...]`90559 **DESCRIPTION**

90560 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall  
 90561 support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD Section 12.2 (on page  
 90562 215).

90563 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell  
 90564 variable specified by the *name* operand and the index of the next argument to be processed in the  
 90565 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

90566 When the option requires an option-argument, the *getopts* utility shall place it in the shell  
 90567 variable *OPTARG*. If no option was found, or if the option that was found does not have an  
 90568 option-argument, *OPTARG* shall be unset.

90569 If an option character not contained in the *optstring* operand is found where an option character  
 90570 is expected, the shell variable specified by *name* shall be set to the <question-mark> ('?')  
 90571 character. In this case, if the first character in *optstring* is a <colon> (':'), the shell variable  
 90572 *OPTARG* shall be set to the option character found, but no output shall be written to standard  
 90573 error; otherwise, the shell variable *OPTARG* shall be unset and a diagnostic message shall be  
 90574 written to standard error. This condition shall be considered to be an error detected in the way  
 90575 arguments were presented to the invoking application, but shall not be an error in *getopts*  
 90576 processing.

90577 If an option-argument is missing:

- 90578 • If the first character of *optstring* is a <colon>, the shell variable specified by *name* shall be  
 90579 set to the <colon> character and the shell variable *OPTARG* shall be set to the option  
 90580 character found.
- 90581 • Otherwise, the shell variable specified by *name* shall be set to the <question-mark>  
 90582 character, the shell variable *OPTARG* shall be unset, and a diagnostic message shall be  
 90583 written to standard error. This condition shall be considered to be an error detected in the  
 90584 way arguments were presented to the invoking application, but shall not be an error in  
 90585 *getopts* processing; a diagnostic message shall be written as stated, but the exit status shall  
 90586 be zero.

90587 When the end of options is encountered, the *getopts* utility shall exit with a return value greater  
 90588 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,  
 90589 where the first "--" argument is considered to be an option-argument if there are no other non-  
 90590 option-arguments appearing before it, or the value "\$#+1" if there are no non-option-  
 90591 arguments; the *name* variable shall be set to the <question-mark> character. Any of the following  
 90592 shall identify the end of options: the special option "--", finding an argument that does not  
 90593 begin with a '-', or encountering an error.

90594 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be  
 90595 exported by default.

90596 The shell variable specified by the *name* operand, *OPTIND*, and *OPTARG* shall affect the current  
 90597 shell execution environment; see Section 2.12 (on page 2331).

90598 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the  
 90599 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple  
 90600 times in a single shell execution environment with parameters (positional parameters or *arg*

|       |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 90601 |                              | operands) that are not the same in all invocations, or with an <i>OPTIND</i> value modified to be a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 90602 |                              | value other than 1, produces unspecified results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 90603 | <b>OPTIONS</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90604 |                              | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 90605 | <b>OPERANDS</b>              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90606 |                              | The following operands shall be supported:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 90607 | <i>optstring</i>             | A string containing the option characters recognized by the utility invoking <i>getopts</i> .<br>If a character is followed by a <colon>, the option shall be expected to have an<br>argument, which should be supplied as a separate argument. Applications should<br>specify an option character and its option-argument as separate arguments, but<br><i>getopts</i> shall interpret the characters following an option character requiring<br>arguments as an argument whether or not this is done. An explicit null option-<br>argument need not be recognized if it is not supplied as a separate argument when<br><i>getopts</i> is invoked. (See also the <i>getopt()</i> function defined in the System Interfaces<br>volume of POSIX.1-2008.) The characters <question-mark> and <colon> shall not<br>be used as option characters by an application. The use of other option characters<br>that are not alphanumeric produces unspecified results. If the option-argument is<br>not supplied as a separate argument from the option character, the value in<br><i>OPTARG</i> shall be stripped of the option character and the ' - '. The first character<br>in <i>optstring</i> determines how <i>getopts</i> behaves if an option character is not known or<br>an option-argument is missing. |
| 90611 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90612 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90613 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90614 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90615 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90616 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90617 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90618 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90619 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90620 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90621 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90622 | <i>name</i>                  | The name of a shell variable that shall be set by the <i>getopts</i> utility to the option<br>character that was found.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 90623 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90624 |                              | The <i>getopts</i> utility by default shall parse positional parameters passed to the invoking shell<br>procedure. If <i>args</i> are given, they shall be parsed instead of the positional parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 90625 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90626 | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90627 |                              | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 90628 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90629 |                              | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 90630 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90631 |                              | The following environment variables shall affect the execution of <i>getopts</i> :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 90632 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null.<br>(See XBD Section 8.2 (on page 174) for the precedence of internationalization<br>variables used to determine the values of locale categories.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 90633 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90634 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90635 | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other<br>internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 90636 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90637 | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as<br>characters (for example, single-byte as opposed to multi-byte characters in<br>arguments and input files).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90638 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90639 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90640 | <i>LC_MESSAGES</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90641 |                              | Determine the locale that should be used to affect the format and contents of<br>diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90642 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90643 | XSI                          | <i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |



90685 **EXAMPLES**

90686 The following example script parses and displays its arguments:

```

90687 aflag=
90688 bflag=
90689 while getopts ab: name
90690 do
90691     case $name in
90692         a)     aflag=1;;
90693         b)     bflag=1
90694             bval="$OPTARG";;
90695         ?)    printf "Usage: %s: [-a] [-b value] args\n" $0
90696             exit 2;;
90697     esac
90698 done
90699 if [ ! -z "$aflag" ]; then
90700     printf "Option -a specified\n"
90701 fi
90702 if [ ! -z "$bflag" ]; then
90703     printf 'Option -b "%s" specified\n' "$bval"
90704 fi
90705 shift $(( $OPTIND - 1 ))
90706 printf "Remaining arguments are: %s\n" "$*"

```

90707 **RATIONALE**

90708 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles  
 90709 option-arguments containing <blank> characters.

90710 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it  
 90711 does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the  
 90712 standard utilities.

90713 The <colon> is not allowed as an option character because that is not historical behavior, and it  
 90714 violates the Utility Syntax Guidelines. The <colon> is now specified to behave as in the  
 90715 KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand,  
 90716 it disables diagnostics concerning missing option-arguments and unexpected option characters.  
 90717 This replaces the use of the *OPTERR* variable that was specified in an early proposal.

90718 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function  
 90719 are not fully specified because implementations with superior (“friendlier”) formats objected to  
 90720 the formats used by some historical implementations. The standard developers considered it  
 90721 important that the information in the messages used be uniform between *getopts* and *getopt()*.  
 90722 Exact duplication of the messages might not be possible, particularly if a utility is built on  
 90723 another system that has a different *getopt()* function, but the messages must have specific  
 90724 information included so that the program name, invalid option character, and type of error can  
 90725 be distinguished by a user.

90726 Only a rare application program intercepts a *getopts* standard error message and wants to parse  
 90727 it. Therefore, implementations are free to choose the most usable messages they can devise. The  
 90728 following formats are used by many historical implementations:

```

90729 "%s: illegal option -- %c\n", <program name>, <option character>
90730 "%s: option requires an argument -- %c\n", <program name>, \
90731     <option character>

```

**getopts***Utilities*

90732 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,  
90733 frequently not even indicating the option character found in error.

90734 **FUTURE DIRECTIONS**

90735 None.

90736 **SEE ALSO**90737 [Section 2.5.2](#) (on page 2302)90738 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)90739 XSH [getopt\(\)](#)90740 **CHANGE HISTORY**

90741 First released in Issue 4.

90742 **Issue 6**

90743 The normative text is reworded to avoid use of the term “must” for application requirements.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

90744 **NAME**90745 `grep` — search a file for a pattern90746 **SYNOPSIS**

```
90747 grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list  
90748 [-e pattern_list]... [-f pattern_file]... [file...]  
90749 grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...  
90750 -f pattern_file [-f pattern_file]... [file...]  
90751 grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]
```

90752 **DESCRIPTION**

90753 The `grep` utility shall search the input files, selecting lines matching one or more patterns; the  
90754 types of patterns are controlled by the options specified. The patterns are specified by the `-e`  
90755 option, `-f` option, or the *pattern\_list* operand. The *pattern\_list*'s value shall consist of one or more  
90756 patterns separated by <newline> characters; the *pattern\_file*'s contents shall consist of one or  
90757 more patterns terminated by a <newline> character. By default, an input line shall be selected if  
90758 any pattern, treated as an entire basic regular expression (BRE) as described in XBD Section 9.3  
90759 (on page 183), matches any part of the line excluding the terminating <newline>; a null BRE  
90760 shall match every line. By default, each selected input line shall be written to the standard  
90761 output.

90762 Regular expression matching shall be based on text lines. Since a <newline> separates or  
90763 terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a  
90764 <newline>. Similarly, since patterns are matched against individual lines (excluding the  
90765 terminating <newline> characters) of the input, there is no way for a pattern to match a  
90766 <newline> found in the input.

90767 **OPTIONS**90768 The `grep` utility shall conform to XBD Section 12.2 (on page 215).

90769 The following options shall be supported:

90770 `-E` Match using extended regular expressions. Treat each pattern specified as an ERE,  
90771 as described in XBD Section 9.4 (on page 188). If any entire ERE pattern matches  
90772 some part of an input line excluding the terminating <newline>, the line shall be  
90773 matched. A null ERE shall match every line.

90774 `-F` Match using fixed strings. Treat each pattern specified as a string instead of a  
90775 regular expression. If an input line contains any of the patterns as a contiguous  
90776 sequence of bytes, the line shall be matched. A null string shall match every line.

90777 `-c` Write only a count of selected lines to standard output.90778 `-e pattern_list`

90779 Specify one or more patterns to be used during the search for input. The  
90780 application shall ensure that patterns in *pattern\_list* are separated by a <newline>.  
90781 A null pattern can be specified by two adjacent <newline> characters in  
90782 *pattern\_list*. Unless the `-E` or `-F` option is also specified, each pattern shall be  
90783 treated as a BRE, as described in XBD Section 9.3 (on page 183). Multiple `-e` and `-f`  
90784 options shall be accepted by the `grep` utility. All of the specified patterns shall be  
90785 used when matching lines, but the order of evaluation is unspecified.

90786 `-f pattern_file`

90787 Read one or more patterns from the file named by the pathname *pattern\_file*.  
90788 Patterns in *pattern\_file* shall be terminated by a <newline>. A null pattern can be  
90789 specified by an empty line in *pattern\_file*. Unless the `-E` or `-F` option is also

**grep**

Utilities

- 90790 specified, each pattern shall be treated as a BRE, as described in XBD [Section 9.3](#)  
 90791 (on page 183).
- 90792 **-i** Perform pattern matching in searches without regard to case; see XBD [Section 9.2](#)  
 90793 (on page 182).
- 90794 **-l** (The letter ell.) Write only the names of files containing selected lines to standard  
 90795 output. Pathnames shall be written once per file searched. If the standard input is  
 90796 searched, a pathname of "(standard input)" shall be written, in the POSIX  
 90797 locale. In other locales, "standard input" may be replaced by something more  
 90798 appropriate in those locales.
- 90799 **-n** Precede each output line by its relative line number in the file, each file starting at  
 90800 line 1. The line number counter shall be reset for each file processed.
- 90801 **-q** Quiet. Nothing shall be written to the standard output, regardless of matching  
 90802 lines. Exit with zero status if an input line is selected.
- 90803 **-s** Suppress the error messages ordinarily written for nonexistent or unreadable files.  
 90804 Other error messages shall not be suppressed.
- 90805 **-v** Select lines not matching any of the specified patterns. If the **-v** option is not  
 90806 specified, selected lines shall be those that match any of the specified patterns.
- 90807 **-x** Consider only input lines that use all characters in the line excluding the  
 90808 terminating <newline> to match an entire fixed string or regular expression to be  
 90809 matching lines.

**OPERANDS**

- 90810 The following operands shall be supported:
- 90811 *pattern\_list* Specify one or more patterns to be used during the search for input. This operand  
 90812 shall be treated as if it were specified as **-e pattern\_list**.  
 90813
- 90814 *file* A pathname of a file to be searched for the patterns. If no *file* operands are  
 90815 specified, the standard input shall be used.

**STDIN**

- 90816 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 90817 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 90818 the standard input shall not be used. See the INPUT FILES section.  
 90819

**INPUT FILES**

- 90820 The input files shall be text files.

**ENVIRONMENT VARIABLES**

- 90821 The following environment variables shall affect the execution of *grep*:
- 90822 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 90823 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 90824 variables used to determine the values of locale categories.)  
 90825
- 90826 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 90827 internationalization variables.  
 90828
- 90829 **LC\_COLLATE**  
 90830 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 90831 character collating elements within regular expressions.

- 90832 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 90833 characters (for example, single-byte as opposed to multi-byte characters in  
 90834 arguments and input files) and the behavior of character classes within regular  
 90835 expressions.
- 90836 **LC\_MESSAGES**  
 90837 Determine the locale that should be used to affect the format and contents of  
 90838 diagnostic messages written to standard error.
- 90839 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 90840 **ASYNCHRONOUS EVENTS**
- 90841 Default.
- 90842 **STDOUT**
- 90843 If the **-l** option is in effect, the following shall be written for each file containing at least one  
 90844 selected input line:
- 90845 "%s\n", <file>
- 90846 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall  
 90847 prefix each output line by:
- 90848 "%s:", <file>
- 90849 The remainder of each output line shall depend on the other options specified:
- 90850 • If the **-c** option is in effect, the remainder of each output line shall contain:
- 90851 "%d\n", <count>
- 90852 • Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written  
 90853 to standard output:
- 90854 "%d:", <line number>
- 90855 • Finally, the following shall be written to standard output:
- 90856 "%s", <selected-line contents>
- 90857 **STDERR**
- 90858 The standard error shall be used only for diagnostic messages.
- 90859 **OUTPUT FILES**
- 90860 None.
- 90861 **EXTENDED DESCRIPTION**
- 90862 None.
- 90863 **EXIT STATUS**
- 90864 The following exit values shall be returned:
- 90865 0 One or more lines were selected.
  - 90866 1 No lines were selected.
  - 90867 >1 An error occurred.
- 90868 **CONSEQUENCES OF ERRORS**
- 90869 If the **-q** option is specified, the exit status shall be zero if an input line is selected, even if an  
 90870 error was detected. Otherwise, default actions shall be performed.

## 90871 APPLICATION USAGE

90872 Care should be taken when using characters in *pattern\_list* that may also be meaningful to the  
 90873 command interpreter. It is safest to enclose the entire *pattern\_list* argument in single-quotes:

90874 ' ... '

90875 The `-e pattern_list` option has the same effect as the *pattern\_list* operand, but is useful when  
 90876 *pattern\_list* begins with the <hyphen> delimiter. It is also useful when it is more convenient to  
 90877 provide multiple patterns as separate arguments.

90878 Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while  
 90879 matching input text lines. (Note that the order of evaluation is not specified. If an  
 90880 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching  
 90881 every line, and effectively ignore any other patterns.)

90882 The `-q` option provides a means of easily determining whether or not a pattern (or string) exists  
 90883 in a group of files. When searching several files, it provides a performance improvement  
 90884 (because it can quit as soon as it finds the first match) and requires less care by the user in  
 90885 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if  
 90886 *grep* detected an access or read error on earlier *file* operands).

## 90887 EXAMPLES

90888 1. To find all uses of the word "Posix" (in any case) in file `text.mm` and write with line  
 90889 numbers:

90890 `grep -i -n posix text.mm`

90891 2. To find all empty lines in the standard input:

90892 `grep ^$`

90893 or:

90894 `grep -v .`

90895 3. Both of the following commands print all lines containing strings "abc" or "def" or  
 90896 both:

90897 `grep -E 'abc|def'`

90898 `grep -F 'abc`  
 90899 `def'`

90900 4. Both of the following commands print all lines matching exactly "abc" or "def":

90901 `grep -E '^abc$|^def$'`

90902 `grep -F -x 'abc`  
 90903 `def'`

## 90904 RATIONALE

90905 This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of  
 90906 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard  
 90907 developers to consolidate the three *greps* into a single command.

90908 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as  
 90909 implementation extensions, allowing historical applications to operate unmodified.

90910 Historical implementations usually silently ignored all but one of multiply-specified `-e` and `-f`  
 90911 options, but were not consistent as to which specification was actually used.

- 90912 The **-b** option was omitted from the OPTIONS section because block numbers are  
90913 implementation-defined.
- 90914 The System V restriction on using **-** to mean standard input was omitted.
- 90915 A definition of action taken when given a null BRE or ERE is specified. This is an error  
90916 condition in some historical implementations.
- 90917 The **-I** option previously indicated that its use was undefined when no files were explicitly  
90918 named. This behavior was historical and placed an unnecessary restriction on future  
90919 implementations. It has been removed.
- 90920 The historical BSD *grep* **-s** option practice is easily duplicated by redirecting standard output to  
90921 **/dev/null**. The **-s** option required here is from System V.
- 90922 The **-x** option, historically available only with *fgrep*, is available here for all of the non-  
90923 obsolescent versions.
- 90924 **FUTURE DIRECTIONS**
- 90925 None.
- 90926 **SEE ALSO**
- 90927 *sed*
- 90928 XBD [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on page 215)
- 90929 **CHANGE HISTORY**
- 90930 First released in Issue 2.
- 90931 **Issue 6**
- 90932 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.
- 90933 The normative text is reworded to avoid use of the term “must” for application requirements.
- 90934 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples  
90935 using the *grep* **-F** option which did not match the normative description of the **-F** option.
- 90936 **Issue 7**
- 90937 Austin Group Interpretation 1003.1-2001 #092 is applied.
- 90938 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 90939 SD5-XCU-ERN-98 is applied, updating the STDOUT section.

**hash**90940 **NAME**

90941 hash — remember or report utility locations

90942 **SYNOPSIS**90943 hash [*utility*...]

90944 hash -r

90945 **DESCRIPTION**

90946 The *hash* utility shall affect the way the current shell environment remembers the locations of  
 90947 utilities found as described in Section 2.9.1.1 (on page 2317). Depending on the arguments  
 90948 specified, it shall add utility locations to its list of remembered locations or it shall purge the  
 90949 contents of the list. When no arguments are specified, it shall report on the contents of the list.

90950 Utilities provided as built-ins to the shell shall not be reported by *hash*.90951 **OPTIONS**90952 The *hash* utility shall conform to XBD Section 12.2 (on page 215).

90953 The following option shall be supported:

90954 -r Forget all previously remembered utility locations

90955 **OPERANDS**

90956 The following operand shall be supported:

90957 *utility* The name of a utility to be searched for and added to the list of remembered  
 90958 locations. If *utility* contains one or more <slash> characters, the results are  
 90959 unspecified.

90960 **STDIN**

90961 Not used.

90962 **INPUT FILES**

90963 None.

90964 **ENVIRONMENT VARIABLES**90965 The following environment variables shall affect the execution of *hash*:

90966 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 90967 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 90968 variables used to determine the values of locale categories.)

90969 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 90970 internationalization variables.

90971 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 90972 characters (for example, single-byte as opposed to multi-byte characters in  
 90973 arguments).

90974 *LC\_MESSAGES*

90975 Determine the locale that should be used to affect the format and contents of  
 90976 diagnostic messages written to standard error.

90977 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.90978 *PATH* Determine the location of *utility*, as described in XBD Chapter 8 (on page 173).

90979 **ASYNCHRONOUS EVENTS**

90980 Default.

90981 **STDOUT**

90982 The standard output of *hash* shall be used when no arguments are specified. Its format is  
 90983 unspecified, but includes the pathname of each utility in the list of remembered locations for the  
 90984 current shell environment. This list shall consist of those utilities named in previous *hash*  
 90985 invocations that have been invoked, and may contain those invoked and found through the  
 90986 normal command search process.

90987 **STDERR**

90988 The standard error shall be used only for diagnostic messages.

90989 **OUTPUT FILES**

90990 None.

90991 **EXTENDED DESCRIPTION**

90992 None.

90993 **EXIT STATUS**

90994 The following exit values shall be returned:

90995 0 Successful completion.

90996 &gt;0 An error occurred.

90997 **CONSEQUENCES OF ERRORS**

90998 Default.

90999 **APPLICATION USAGE**

91000 Since *hash* affects the current shell execution environment, it is always provided as a shell  
 91001 regular built-in. If it is called in a separate utility execution environment, such as one of the  
 91002 following:

```
91003 nohup hash -r
91004 find . -type f | xargs hash
```

91005 it does not affect the command search process of the caller's environment.

91006 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities  
 91007 found through normal command search are not listed by the *hash* command.

91008 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the  
 91009 simplest form, this can be:

91010 `PATH="$PATH"`

91011 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a  
 91012 performance improvement on a few implementations; normally, the hashing process is included  
 91013 by default.

91014 **EXAMPLES**

91015 None.

91016 **RATIONALE**

91017 None.

**hash***Utilities*91018 **FUTURE DIRECTIONS**

91019 None.

91020 **SEE ALSO**91021 [Section 2.9.1.1](#) (on page 2317)91022 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)91023 **CHANGE HISTORY**

91024 First released in Issue 2.

91025 **Issue 7**91026 The *hash* utility is moved from the XSI option to the Base.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

91027 **NAME**

91028 head — copy the first part of files

91029 **SYNOPSIS**91030 head [-n *number*] [*file...*]91031 **DESCRIPTION**91032 The *head* utility shall copy its input files to the standard output, ending the output for each file at  
91033 a designated point.91034 Copying shall end at the point in each input file indicated by the **-n** *number* option. The option-  
91035 argument *number* shall be counted in units of lines.91036 **OPTIONS**91037 The *head* utility shall conform to XBD Section 12.2 (on page 215).

91038 The following option shall be supported:

91039 **-n** *number* The first *number* lines of each input file shall be copied to standard output. The  
91040 application shall ensure that the *number* option-argument is a positive decimal  
91041 integer.91042 When a file contains less than *number* lines, it shall be copied to standard output in its entirety.  
91043 This shall not be an error.91044 If no options are specified, *head* shall act as if **-n 10** had been specified.91045 **OPERANDS**

91046 The following operand shall be supported:

91047 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
91048 shall be used.91049 **STDIN**91050 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
91051 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
91052 the standard input shall not be used. See the INPUT FILES section.91053 **INPUT FILES**

91054 Input files shall be text files, but the line length is not restricted to {LINE\_MAX} bytes.

91055 **ENVIRONMENT VARIABLES**91056 The following environment variables shall affect the execution of *head*:91057 **LANG** Provide a default value for the internationalization variables that are unset or null.  
91058 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
91059 variables used to determine the values of locale categories.)91060 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
91061 internationalization variables.91062 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
91063 characters (for example, single-byte as opposed to multi-byte characters in  
91064 arguments and input files).91065 **LC\_MESSAGES**91066 Determine the locale that should be used to affect the format and contents of  
91067 diagnostic messages written to standard error.

**head**

Utilities

91068 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

91069 **ASYNCHRONOUS EVENTS**

91070 Default.

91071 **STDOUT**

91072 The standard output shall contain designated portions of the input files.

91073 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

91074 "`\n==> %s <==\n`", *<pathname>*

91075 except that the first header written shall not include the initial *<newline>*.

91076 **STDERR**

91077 The standard error shall be used only for diagnostic messages.

91078 **OUTPUT FILES**

91079 None.

91080 **EXTENDED DESCRIPTION**

91081 None.

91082 **EXIT STATUS**

91083 The following exit values shall be returned:

91084 0 Successful completion.

91085 >0 An error occurred.

91086 **CONSEQUENCES OF ERRORS**

91087 Default.

91088 **APPLICATION USAGE**

91089 None.

91090 **EXAMPLES**

91091 To write the first ten lines of all files (except those with a leading period) in the directory:

91092 `head -- *`

91093 **RATIONALE**

91094 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside *tail*.

91097 POSIX.1-2008 version of *head* follows the Utility Syntax Guidelines. The `-n` option was added to this new interface so that *head* and *tail* would be more logically related. Earlier versions of this standard allowed a `-number` option. This form is no longer specified by POSIX.1-2008 but may be present in some implementations.

91101 There is no `-c` option (as there is in *tail*) because it is not historical practice and because other utilities in this volume of POSIX.1-2008 provide similar functionality.

91103 **FUTURE DIRECTIONS**

91104 None.

91105 **SEE ALSO**

91106 *sed*, *tail*

91107 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

91108 **CHANGE HISTORY**

91109 First released in Issue 4.

91110 **Issue 6**91111 The obsolescent **–number** form is removed.

91112 The normative text is reworded to avoid use of the term “must” for application requirements.

91113 The DESCRIPTION is updated to clarify that when a file contains less than the number of lines  
91114 requested, the entire file is copied to standard output.91115 **Issue 7**

91116 Austin Group Interpretations 1003.1-2001 #027 and #092 are applied.

91117 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91118 The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**iconv**

Utilities

91119 **NAME**

91120 iconv — codeset conversion

91121 **SYNOPSIS**91122 iconv [-cs] -f *frommap* -t *tomap* [*file...*]91123 iconv -f *fromcode* [-cs] [-t *tocode*] [*file...*]91124 iconv -t *tocode* [-cs] [-f *fromcode*] [*file...*]

91125 iconv -l

91126 **DESCRIPTION**91127 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and  
91128 write the results to standard output.91129 When the options indicate that charmap files are used to specify the codesets (see OPTIONS),  
91130 the codeset conversion shall be accomplished by performing a logical join on the symbolic  
91131 character names in the two charmaps. The implementation need not support the use of charmap  
91132 files for codeset conversion unless the POSIX2\_LOCALEDEF symbol is defined on the system.91133 **OPTIONS**91134 The *iconv* utility shall conform to XBD Section 12.2 (on page 215).

91135 The following options shall be supported:

91136 **-c** Omit any characters that are invalid in the codeset of the input file from the  
91137 output. When **-c** is not used, the results of encountering invalid characters in the  
91138 input stream (either those that are not characters in the codeset of the input file or  
91139 that have no corresponding character in the codeset of the output file) shall be  
91140 specified in the system documentation. The presence or absence of **-c** shall not  
91141 affect the exit status of *iconv*.91142 **-f** *fromcodeset*91143 Identify the codeset of the input file. The implementation shall recognize the  
91144 following two forms of the *fromcodeset* option-argument:91145 *fromcode* The *fromcode* option-argument must not contain a <slash> character.  
91146 It shall be interpreted as the name of one of the codeset descriptions  
91147 provided by the implementation in an unspecified format. Valid  
91148 values of *fromcode* are implementation-defined.91149 *frommap*91150 The *frommap* option-argument must contain a <slash> character. It  
91151 shall be interpreted as the pathname of a charmap file as defined in  
91152 XBD Section 6.4 (on page 129). If the pathname does not represent a  
valid, readable charmap file, the results are undefined.

91153 If this option is omitted, the codeset of the current locale shall be used.

91154 **-l**91155 Write all supported *fromcode* and *tocode* values to standard output in an unspecified  
format.91156 **-s**91157 Suppress any messages written to standard error concerning invalid characters.  
91158 When **-s** is not used, the results of encountering invalid characters in the input  
91159 stream (either those that are not valid characters in the codeset of the input file or  
91160 that have no corresponding character in the codeset of the output file) shall be  
91161 specified in the system documentation. The presence or absence of **-s** shall not  
affect the exit status of *iconv*.



91202 **OUTPUT FILES**

91203 None.

91204 **EXTENDED DESCRIPTION**

91205 None.

91206 **EXIT STATUS**

91207 The following exit values shall be returned:

91208 0 Successful completion.

91209 &gt;0 An error occurred.

91210 **CONSEQUENCES OF ERRORS**

91211 Default.

91212 **APPLICATION USAGE**91213 The user must ensure that both charmap files use the same symbolic names for characters the  
91214 two codesets have in common.91215 **EXAMPLES**91216 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001  
91217 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file  
91218 **mail.local**:91219 `iconv -f IS6937 -t IS8859 mail.x400 > mail.local`91220 **RATIONALE**91221 The *iconv* utility can be used portably only when the user provides two charmap files as option-  
91222 arguments. This is because a single charmap provided by the user cannot reliably be joined with  
91223 the names in a system-provided character set description. The valid values for *fromcode* and  
91224 *tocode* are implementation-defined and do not have to have any relation to the charmap  
91225 mechanisms. As an aid to interactive users, the `-I` option was adopted from the Plan 9 operating  
91226 system. It writes information concerning these implementation-defined values. The format is  
91227 unspecified because there are many possible useful formats that could be chosen, such as a  
91228 matrix of valid combinations of *fromcode* and *tocode*. The `-I` option is not intended for shell script  
91229 usage; conforming applications will have to use charmaps.91230 **FUTURE DIRECTIONS**

91231 None.

91232 **SEE ALSO**91233 *gencat*91234 XBD [Section 6.4](#) (on page 129), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)91235 **CHANGE HISTORY**

91236 First released in Issue 3.

91237 **Issue 6**91238 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the  
91239 ability to use charmap files for conversion has been added.91240 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address  
91241 inconsistencies with the *iconv()* function in the System Interfaces volume of POSIX.1-2008.

91242 **Issue 7**91243 Austin Group Interpretation 1003.1-2001 #206 is applied, correcting the *tomap* option.

91244 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**id**

Utilities

91245 **NAME**91246 **id** — return user identity91247 **SYNOPSIS**91248 **id** [*user*]91249 **id** -G [-n] [*user*]91250 **id** -g [-nr] [*user*]91251 **id** -u [-nr] [*user*]91252 **DESCRIPTION**

91253 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the  
 91254 corresponding user and group names of the invoking process to standard output. If the effective  
 91255 and real IDs do not match, both shall be written. If multiple groups are supported by the  
 91256 underlying system (see the description of {NGROUPS\_MAX} in the System Interfaces volume of  
 91257 POSIX.1-2008), the supplementary group affiliations of the invoking process shall also be  
 91258 written.

91259 If a *user* operand is provided and the process has appropriate privileges, the user and group IDs  
 91260 of the selected user shall be written. In this case, effective IDs shall be assumed to be identical to  
 91261 real IDs. If the selected user has more than one allowable group membership listed in the group  
 91262 database, these shall be written in the same manner as the supplementary groups described in  
 91263 the preceding paragraph.

91264 **OPTIONS**91265 The *id* utility shall conform to XBD Section 12.2 (on page 215).

91266 The following options shall be supported:

91267 **-G** Output all different group IDs (effective, real, and supplementary) only, using the  
 91268 format "%u\n". If there is more than one distinct group affiliation, output each  
 91269 such affiliation, using the format " %u", before the <newline> is output.

91270 **-g** Output only the effective group ID, using the format "%u\n".

91271 **-n** Output the name in the format "%s" instead of the numeric ID using the format  
 91272 "%u".

91273 **-r** Output the real ID instead of the effective ID.

91274 **-u** Output only the effective user ID, using the format "%u\n".

91275 **OPERANDS**

91276 The following operand shall be supported:

91277 *user* The login name for which information is to be written.

91278 **STDIN**

91279 Not used.

91280 **INPUT FILES**

91281 None.

91282 **ENVIRONMENT VARIABLES**91283 The following environment variables shall affect the execution of *id*:

91284 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 91285 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 91286 variables used to determine the values of locale categories.)

- 91287 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
91288 internationalization variables.
- 91289 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
91290 characters (for example, single-byte as opposed to multi-byte characters in  
91291 arguments).
- 91292 *LC\_MESSAGES*  
91293 Determine the locale that should be used to affect the format and contents of  
91294 diagnostic messages written to standard error and informative messages written to  
91295 standard output.
- 91296 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 91297 **ASYNCHRONOUS EVENTS**  
91298 Default.
- 91299 **STDOUT**  
91300 The following formats shall be used when the *LC\_MESSAGES* locale category specifies the  
91301 POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with  
91302 more appropriate strings corresponding to the locale.
- 91303 "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,  
91304 <real group ID>, <group-name>
- 91305 If the effective and real user IDs do not match, the following shall be inserted immediately  
91306 before the '\n' character in the previous format:
- 91307 " euid=%u(%s) "
- 91308 with the following arguments added at the end of the argument list:  
91309 <effective user ID>, <effective user-name>
- 91310 If the effective and real group IDs do not match, the following shall be inserted directly before  
91311 the '\n' character in the format string (and after any addition resulting from the effective and  
91312 real user IDs not matching):
- 91313 " egid=%u(%s) "
- 91314 with the following arguments added at the end of the argument list:  
91315 <effective group-ID>, <effective group name>
- 91316 If the process has supplementary group affiliations or the selected user is allowed to belong to  
91317 multiple groups, the first shall be added directly before the <newline> in the format string:
- 91318 " groups=%u(%s) "
- 91319 with the following arguments added at the end of the argument list:  
91320 <supplementary group ID>, <supplementary group name>
- 91321 and the necessary number of the following added after that for any remaining supplementary  
91322 group IDs:
- 91323 ", %u(%s) "
- 91324 and the necessary number of the following arguments added at the end of the argument list:  
91325 <supplementary group ID>, <supplementary group name>
- 91326 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple

91327 group IDs cannot be mapped by the system into printable user or group names, the  
 91328 corresponding "(%s)" and *name* argument shall be omitted from the corresponding format  
 91329 string.

91330 When any of the options are specified, the output format shall be as described in the OPTIONS  
 91331 section.

#### 91332 **STDERR**

91333 The standard error shall be used only for diagnostic messages.

#### 91334 **OUTPUT FILES**

91335 None.

#### 91336 **EXTENDED DESCRIPTION**

91337 None.

#### 91338 **EXIT STATUS**

91339 The following exit values shall be returned:

91340 0 Successful completion.

91341 >0 An error occurred.

#### 91342 **CONSEQUENCES OF ERRORS**

91343 Default.

#### 91344 **APPLICATION USAGE**

91345 Output produced by the **-G** option and by the default case could potentially produce very long  
 91346 lines on systems that support large numbers of supplementary groups. (On systems with user  
 91347 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per  
 91348 name, 93 supplementary groups plus distinct effective and real group and user IDs could  
 91349 theoretically overflow the 2048-byte {LINE\_MAX} text file line limit on the default output case.  
 91350 It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id -G*.)  
 91351 This is not expected to be a problem in practice, but in cases where it is a concern, applications  
 91352 should consider using *fold -s* before post-processing the output of *id*.

#### 91353 **EXAMPLES**

91354 None.

#### 91355 **RATIONALE**

91356 The functionality provided by the 4 BSD *groups* utility can be simulated using:

91357 `id -Gn [ user ]`

91358 The 4 BSD command *groups* was considered, but it was not included because it did not provide  
 91359 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to  
 91360 modify *id* to provide the additional functionality necessary to systems with multiple groups than  
 91361 to invent another command.

91362 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands  
 91363 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select  
 91364 the desired piece of information. Since output such as that produced by:

91365 `id -u -n`

91366 is frequently wanted, it seemed desirable to add the options.

91367 **FUTURE DIRECTIONS**

91368 None.

91369 **SEE ALSO**91370 *fold, logname, who*91371 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)91372 XSH [getgid\(\)](#), [getgroups\(\)](#), [getuid\(\)](#)91373 **CHANGE HISTORY**

91374 First released in Issue 2.

91375 **Issue 7**

91376 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**ipcrm**

Utilities

91377 **NAME**91378 `ipcrm` — remove an XSI message queue, semaphore set, or shared memory segment identifier91379 **SYNOPSIS**91380 XSI `ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey] ...`91381 **DESCRIPTION**91382 The `ipcrm` utility shall remove zero or more message queues, semaphore sets, or shared memory  
91383 segments. The interprocess communication facilities to be removed are specified by the options.91384 Only a user with appropriate privileges shall be allowed to remove an interprocess  
91385 communication facility that was not created by or owned by the user invoking `ipcrm`.91386 **OPTIONS**91387 The `ipcrm` utility shall conform to XBD [Section 12.2](#) (on page 215).

91388 The following options shall be supported:

- 91389 `-q msgid` Remove the message queue identifier `msgid` from the system and destroy the  
91390 message queue and data structure associated with it.
- 91391 `-m shmid` Remove the shared memory identifier `shmid` from the system. The shared memory  
91392 segment and data structure associated with it shall be destroyed after the last  
91393 detach.
- 91394 `-s semid` Remove the semaphore identifier `semid` from the system and destroy the set of  
91395 semaphores and data structure associated with it.
- 91396 `-Q msgkey` Remove the message queue identifier, created with key `msgkey`, from the system  
91397 and destroy the message queue and data structure associated with it.
- 91398 `-M shmkey` Remove the shared memory identifier, created with key `shmkey`, from the system.  
91399 The shared memory segment and data structure associated with it shall be  
91400 destroyed after the last detach.
- 91401 `-S semkey` Remove the semaphore identifier, created with key `semkey`, from the system and  
91402 destroy the set of semaphores and data structure associated with it.

91403 **OPERANDS**

91404 None.

91405 **STDIN**

91406 Not used.

91407 **INPUT FILES**

91408 None.

91409 **ENVIRONMENT VARIABLES**91410 The following environment variables shall affect the execution of `ipcrm`:

- 91411 `LANG` Provide a default value for the internationalization variables that are unset or null.  
91412 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
91413 variables used to determine the values of locale categories.)
- 91414 `LC_ALL` If set to a non-empty string value, override the values of all the other  
91415 internationalization variables.
- 91416 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
91417 characters (for example, single-byte as opposed to multi-byte characters in  
91418 arguments).

- 91419 *LC\_MESSAGES*
- 91420 Determine the locale that should be used to affect the format and contents of
- 91421 diagnostic messages written to standard error.
- 91422 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 91423 **ASYNCHRONOUS EVENTS**
- 91424 Default.
- 91425 **STDOUT**
- 91426 Not used.
- 91427 **STDERR**
- 91428 The standard error shall be used only for diagnostic messages.
- 91429 **OUTPUT FILES**
- 91430 None.
- 91431 **EXTENDED DESCRIPTION**
- 91432 None.
- 91433 **EXIT STATUS**
- 91434 The following exit values shall be returned:
- 91435 0 Successful completion.
- 91436 >0 An error occurred.
- 91437 **CONSEQUENCES OF ERRORS**
- 91438 Default.
- 91439 **APPLICATION USAGE**
- 91440 None.
- 91441 **EXAMPLES**
- 91442 None.
- 91443 **RATIONALE**
- 91444 None.
- 91445 **FUTURE DIRECTIONS**
- 91446 None.
- 91447 **SEE ALSO**
- 91448 *ipcs*
- 91449 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
- 91450 XSH *msgctl()*, *semctl()*, *shmctl()*
- 91451 **CHANGE HISTORY**
- 91452 First released in Issue 5.
- 91453 **Issue 7**
- 91454 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**ipcs**

Utilities

91455 **NAME**91456 `ipcs` — report XSI interprocess communication facilities status91457 **SYNOPSIS**91458 XSI `ipcs [-qms] [-a|-bcopt]`91459 **DESCRIPTION**91460 The *ipcs* utility shall write information about active interprocess communication facilities.

91461 Without options, information shall be written in short format for message queues, shared  
 91462 memory segments, and semaphore sets that are currently active in the system. Otherwise, the  
 91463 information that is displayed is controlled by the options specified.

91464 **OPTIONS**91465 The *ipcs* utility shall conform to XBD Section 12.2 (on page 215).91466 The *ipcs* utility accepts the following options:91467 **-q** Write information about active message queues.91468 **-m** Write information about active shared memory segments.91469 **-s** Write information about active semaphore sets.

91470 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of  
 91471 these three are specified, information about all three shall be written subject to the following  
 91472 options:

91473 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

91474 **-b** Write information on maximum allowable size. (Maximum number of bytes in  
 91475 messages on queue for message queues, size of segments for shared memory, and  
 91476 number of semaphores in each set for semaphores.)

91477 **-c** Write creator's user name and group name; see below.

91478 **-o** Write information on outstanding usage. (Number of messages on queue and total  
 91479 number of bytes in messages on queue for message queues, and number of  
 91480 processes attached to shared memory segments.)

91481 **-p** Write process number information. (Process ID of the last process to send a  
 91482 message and process ID of the last process to receive a message on message  
 91483 queues, process ID of the creating process, and process ID of the last process to  
 91484 attach or detach on shared memory segments.)

91485 **-t** Write time information. (Time of the last control operation that changed the access  
 91486 permissions for all facilities, time of the last *msgsnd()* and *msgrcv()* operations on  
 91487 message queues, time of the last *shmat()* and *shmdt()* operations on shared  
 91488 memory, and time of the last *semop()* operation on semaphores.)

91489 **OPERANDS**

91490 None.

91491 **STDIN**

91492 Not used.

91493 **INPUT FILES**

- 91494 • The group database

- 91495 • The user database

#### 91496 ENVIRONMENT VARIABLES

91497 The following environment variables shall affect the execution of *ipcs*:

- 91498 *LANG* Provide a default value for the internationalization variables that are unset or null.  
91499 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
91500 variables used to determine the values of locale categories.)
- 91501 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
91502 internationalization variables.
- 91503 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
91504 characters (for example, single-byte as opposed to multi-byte characters in  
91505 arguments).
- 91506 *LC\_MESSAGES*  
91507 Determine the locale that should be used to affect the format and contents of  
91508 diagnostic messages written to standard error.
- 91509 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 91510 *TZ* Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset  
91511 or null, an unspecified default timezone shall be used.

#### 91512 ASYNCHRONOUS EVENTS

91513 Default.

#### 91514 STDOUT

91515 An introductory line shall be written with the format:

91516 "IPC status from %s as of %s\n", <source>, <date>

91517 where <source> indicates the source used to gather the statistics and <date> is the information  
91518 that would be produced by the *date* command when invoked in the POSIX locale.

91519 The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options.  
91520 The first report shall indicate the status of message queues, the second report shall indicate the  
91521 status of shared memory segments, and the third report shall indicate the status of semaphore  
91522 sets.

91523 If the corresponding facility is not installed or has not been used since the last reboot, then the  
91524 report shall be written out in the format:

91525 "%s facility not in system.\n", <facility>

91526 where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has  
91527 been installed and has been used since the last reboot, column headings separated by one or  
91528 more <space> characters and followed by a <newline> shall be written as indicated below  
91529 followed by the facility name written out using the format:

91530 "%s:\n", <facility>

91531 where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second  
91532 and third reports the column headings need not be written if the last column headings written  
91533 already provide column headings for all information in that report.

91534 The column headings provided in the first column below and the meaning of the information in  
91535 those columns shall be given in order below; the letters in parentheses indicate the options that  
91536 shall cause the corresponding column to appear; "all" means that the column shall always  
91537 appear. Each column is separated by one or more <space> characters. Note that these options

|       |            |                                                                                                                                                                                                                                                                                                                                                         |
|-------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 91538 |            | only determine what information is provided for each report; they do not determine which reports are written.                                                                                                                                                                                                                                           |
| 91539 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91540 | T (all)    | Type of facility:                                                                                                                                                                                                                                                                                                                                       |
| 91541 |            | q Message queue.                                                                                                                                                                                                                                                                                                                                        |
| 91542 |            | m Shared memory segment.                                                                                                                                                                                                                                                                                                                                |
| 91543 |            | s Semaphore.                                                                                                                                                                                                                                                                                                                                            |
| 91544 |            | This field is a single character written using the format %c.                                                                                                                                                                                                                                                                                           |
| 91545 | ID (all)   | The identifier for the facility entry. This field shall be written using the format %d.                                                                                                                                                                                                                                                                 |
| 91546 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91547 | KEY (all)  | The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the facility entry.                                                                                                                                                                                                                                     |
| 91548 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91549 |            | <b>Note:</b> The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.                                                                                                                                                                                      |
| 91550 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91551 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91552 |            | This field shall be written using the format 0x%x.                                                                                                                                                                                                                                                                                                      |
| 91553 | MODE (all) | The facility access modes and flags. The mode shall consist of 11 characters that are interpreted as follows.                                                                                                                                                                                                                                           |
| 91554 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91555 |            | The first character shall be:                                                                                                                                                                                                                                                                                                                           |
| 91556 |            | S If a process is waiting on a <i>msgsnd()</i> operation.                                                                                                                                                                                                                                                                                               |
| 91557 |            | – If the above is not true.                                                                                                                                                                                                                                                                                                                             |
| 91558 |            | The second character shall be:                                                                                                                                                                                                                                                                                                                          |
| 91559 |            | R If a process is waiting on a <i>msgrcv()</i> operation.                                                                                                                                                                                                                                                                                               |
| 91560 |            | C or – If the associated shared memory segment is to be cleared when the first attach operation is executed.                                                                                                                                                                                                                                            |
| 91561 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91562 |            | – If none of the above is true.                                                                                                                                                                                                                                                                                                                         |
| 91563 |            | The next nine characters shall be interpreted as three sets of three bits each.                                                                                                                                                                                                                                                                         |
| 91564 |            | The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is a minus-sign ('-'). |
| 91565 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91566 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91567 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91568 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91569 |            | The permissions shall be indicated as follows:                                                                                                                                                                                                                                                                                                          |
| 91570 |            | r If read permission is granted.                                                                                                                                                                                                                                                                                                                        |
| 91571 |            | w If write permission is granted.                                                                                                                                                                                                                                                                                                                       |
| 91572 |            | a If alter permission is granted.                                                                                                                                                                                                                                                                                                                       |
| 91573 |            | – If the indicated permission is not granted.                                                                                                                                                                                                                                                                                                           |
| 91574 |            | The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> shall be written; otherwise, another printable character is                              |
| 91575 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91576 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 91577 |            |                                                                                                                                                                                                                                                                                                                                                         |

|       |               |                                                                                                                                                                                                                                                                                                               |
|-------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 91578 |               | written.                                                                                                                                                                                                                                                                                                      |
| 91579 | OWNER (all)   | The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the owner shall be written using the format %d.                          |
| 91580 |               |                                                                                                                                                                                                                                                                                                               |
| 91581 |               |                                                                                                                                                                                                                                                                                                               |
| 91582 |               |                                                                                                                                                                                                                                                                                                               |
| 91583 | GROUP (all)   | The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the owner shall be written using the format %d.                      |
| 91584 |               |                                                                                                                                                                                                                                                                                                               |
| 91585 |               |                                                                                                                                                                                                                                                                                                               |
| 91586 |               |                                                                                                                                                                                                                                                                                                               |
| 91587 |               | The following nine columns shall be only written out for message queues:                                                                                                                                                                                                                                      |
| 91588 | CREATOR (a,c) | The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.                    |
| 91589 |               |                                                                                                                                                                                                                                                                                                               |
| 91590 |               |                                                                                                                                                                                                                                                                                                               |
| 91591 |               |                                                                                                                                                                                                                                                                                                               |
| 91592 | CGROUP (a,c)  | The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.                |
| 91593 |               |                                                                                                                                                                                                                                                                                                               |
| 91594 |               |                                                                                                                                                                                                                                                                                                               |
| 91595 |               |                                                                                                                                                                                                                                                                                                               |
| 91596 | CBYTES (a,o)  | The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                       |
| 91597 |               |                                                                                                                                                                                                                                                                                                               |
| 91598 | QNUM (a,o)    | The number of messages currently outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                                |
| 91599 |               |                                                                                                                                                                                                                                                                                                               |
| 91600 | QBYTES (a,b)  | The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                 |
| 91601 |               |                                                                                                                                                                                                                                                                                                               |
| 91602 | LSPID (a,p)   | The process ID of the last process to send a message to the associated queue. This field shall be written using the format:                                                                                                                                                                                   |
| 91603 |               |                                                                                                                                                                                                                                                                                                               |
| 91604 |               | "%d", <pid>                                                                                                                                                                                                                                                                                                   |
| 91605 |               | where <pid> is 0 if no message has been sent to the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to send a message to the queue.                                                                                                                                 |
| 91606 |               |                                                                                                                                                                                                                                                                                                               |
| 91607 |               |                                                                                                                                                                                                                                                                                                               |
| 91608 | LRPID (a,p)   | The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:                                                                                                                                                                              |
| 91609 |               |                                                                                                                                                                                                                                                                                                               |
| 91610 |               | "%d", <pid>                                                                                                                                                                                                                                                                                                   |
| 91611 |               | where <pid> is 0 if no message has been received from the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to receive a message from the queue.                                                                                                                      |
| 91612 |               |                                                                                                                                                                                                                                                                                                               |
| 91613 |               |                                                                                                                                                                                                                                                                                                               |
| 91614 | STIME (a,t)   | The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be written. |
| 91615 |               |                                                                                                                                                                                                                                                                                                               |
| 91616 |               |                                                                                                                                                                                                                                                                                                               |
| 91617 |               |                                                                                                                                                                                                                                                                                                               |
| 91618 |               |                                                                                                                                                                                                                                                                                                               |
| 91619 | RTIME (a,t)   | The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue                                                                                    |
| 91620 |               |                                                                                                                                                                                                                                                                                                               |
| 91621 |               |                                                                                                                                                                                                                                                                                                               |

|       |               |                                                                                   |
|-------|---------------|-----------------------------------------------------------------------------------|
| 91622 |               | shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format           |
| 91623 |               | " no-entry" shall be written.                                                     |
| 91624 |               | The following eight columns shall be only written out for shared memory segments. |
| 91625 | CREATOR (a,c) | The user of the creator of the facility entry. If the user name of the creator is |
| 91626 |               | found in the user database, at least the first eight column positions of the      |
| 91627 |               | name shall be written using the format %s. Otherwise, the user ID of the          |
| 91628 |               | creator shall be written using the format %d.                                     |
| 91629 | CGROUP (a,c)  | The group name of the creator of the facility entry. If the group name of the     |
| 91630 |               | creator is found in the group database, at least the first eight column positions |
| 91631 |               | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 91632 |               | the creator shall be written using the format %d.                                 |
| 91633 | NATTCH (a,o)  | The number of processes attached to the associated shared memory segment.         |
| 91634 |               | This field shall be written using the format %d.                                  |
| 91635 | SEGSZ (a,b)   | The size of the associated shared memory segment. This field shall be written     |
| 91636 |               | using the format %d.                                                              |
| 91637 | CPID (a,p)    | The process ID of the creator of the shared memory entry. This field shall be     |
| 91638 |               | written using the format %d.                                                      |
| 91639 | LPID (a,p)    | The process ID of the last process to attach or detach the shared memory          |
| 91640 |               | segment. This field shall be written using the format:                            |
| 91641 |               | "%d", <pid>                                                                       |
| 91642 |               | where <pid> is 0 if no process has attached the corresponding shared memory       |
| 91643 |               | segment; otherwise, <pid> shall be the process ID of the last process to attach   |
| 91644 |               | or detach the segment.                                                            |
| 91645 | ATIME (a,t)   | The time the last attach on the associated shared memory segment was              |
| 91646 |               | completed. If the corresponding shared memory segment has ever been               |
| 91647 |               | attached, the hour, minute, and second of the last time the segment was           |
| 91648 |               | attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the         |
| 91649 |               | format " no-entry" shall be written.                                              |
| 91650 | DTIME (a,t)   | The time the last detach on the associated shared memory segment was              |
| 91651 |               | completed. If the corresponding shared memory segment has ever been               |
| 91652 |               | detached, the hour, minute, and second of the last time the segment was           |
| 91653 |               | detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the         |
| 91654 |               | format " no-entry" shall be written.                                              |
| 91655 |               | The following four columns shall be only written out for semaphore sets:          |
| 91656 | CREATOR (a,c) | The user of the creator of the facility entry. If the user name of the creator is |
| 91657 |               | found in the user database, at least the first eight column positions of the      |
| 91658 |               | name shall be written using the format %s. Otherwise, the user ID of the          |
| 91659 |               | creator shall be written using the format %d.                                     |
| 91660 | CGROUP (a,c)  | The group name of the creator of the facility entry. If the group name of the     |
| 91661 |               | creator is found in the group database, at least the first eight column positions |
| 91662 |               | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 91663 |               | the creator shall be written using the format %d.                                 |

- 91664 NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.  
91665 This field shall be written using the format %d.
- 91666 OTIME (a,t) The time the last semaphore operation on the set associated with the  
91667 semaphore entry was completed. If a semaphore operation has ever been  
91668 performed on the corresponding semaphore set, the hour, minute, and second  
91669 of the last semaphore operation on the semaphore set shall be written using  
91670 the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be  
91671 written.
- 91672 The following column shall be written for all three reports when it is requested:
- 91673 CTIME (a,t) The time the associated entry was created or changed. The hour, minute, and  
91674 second of the time when the associated entry was created shall be written  
91675 using the format %d:%2.2d:%2.2d.
- 91676 **STDERR**  
91677 The standard error shall be used only for diagnostic messages.
- 91678 **OUTPUT FILES**  
91679 None.
- 91680 **EXTENDED DESCRIPTION**  
91681 None.
- 91682 **EXIT STATUS**  
91683 The following exit values shall be returned:  
91684 0 Successful completion.  
91685 >0 An error occurred.
- 91686 **CONSEQUENCES OF ERRORS**  
91687 Default.
- 91688 **APPLICATION USAGE**  
91689 Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate  
91690 only when it was retrieved.
- 91691 **EXAMPLES**  
91692 None.
- 91693 **RATIONALE**  
91694 None.
- 91695 **FUTURE DIRECTIONS**  
91696 None.
- 91697 **SEE ALSO**  
91698 *ipcrm*  
91699 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)  
91700 XSH *msgrcv()*, *msgsnd()*, *semget()*, *semop()*, *shmat()*, *shmdt()*, *shmget()*
- 91701 **CHANGE HISTORY**  
91702 First released in Issue 5.

91703 **Issue 6**

91704 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

91705 The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

91706 The Open Group Base Resolution bwg98-004 is applied.

91707 **Issue 7**

91708 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91709 SD5-XCU-ERN-139 is applied, adding the *ipcrm* utility to the SEE ALSO section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

91710 **NAME**

91711 jobs — display status of jobs in the current session

91712 **SYNOPSIS**91713 UP jobs [-l|-p] [*job\_id*...]91714 **DESCRIPTION**91715 The *jobs* utility shall display the status of jobs that were started in the current shell environment;  
91716 see Section 2.12 (on page 2331).91717 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the  
91718 list of those “known in the current shell execution environment”; see Section 2.9.3.1 (on page  
91719 2319).91720 **OPTIONS**91721 The *jobs* utility shall conform to XBD Section 12.2 (on page 215).

91722 The following options shall be supported:

91723 **-l** (The letter ell.) Provide more information about each job listed. This information  
91724 shall include the job number, current job, process group ID, state, and the  
91725 command that formed the job.91726 **-p** Display only the process IDs for the process group leaders of the selected jobs.91727 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs  
91728 and all jobs whose status has changed and have not been reported by the shell.91729 **OPERANDS**

91730 The following operand shall be supported:

91731 *job\_id* Specifies the jobs for which the status is to be displayed. If no *job\_id* is given, the  
91732 status information for all jobs shall be displayed. The format of *job\_id* is described  
91733 in XBD Section 3.203 (on page 65).91734 **STDIN**

91735 Not used.

91736 **INPUT FILES**

91737 None.

91738 **ENVIRONMENT VARIABLES**91739 The following environment variables shall affect the execution of *jobs*:91740 **LANG** Provide a default value for the internationalization variables that are unset or null.  
91741 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
91742 variables used to determine the values of locale categories.)91743 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
91744 internationalization variables.91745 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
91746 characters (for example, single-byte as opposed to multi-byte characters in  
91747 arguments).91748 **LC\_MESSAGES**91749 Determine the locale that should be used to affect the format and contents of  
91750 diagnostic messages written to standard error and informative messages written to  
91751 standard output.

- 91752 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 91753 **ASYNCHRONOUS EVENTS**
- 91754 Default.
- 91755 **STDOUT**
- 91756 If the **-p** option is specified, the output shall consist of one line for each process ID:
- 91757 "%d\n", <process ID>
- 91758 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:
- 91759 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>
- 91760 where the fields shall be as follows:
- 91761 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*  
 91762 utilities; this job can also be specified using the *job\_id* %+ or "%%". The character  
 91763 '- ' identifies the job that would become the default if the current default job were  
 91764 to exit; this job can also be specified using the *job\_id* %-. For other jobs, this field is  
 91765 a <space>. At most one job can be identified with '+' and at most one job can be  
 91766 identified with '- '. If there is any suspended job, then the current job shall be a  
 91767 suspended job. If there are at least two suspended jobs, then the previous job also  
 91768 shall be a suspended job.
- 91769 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*  
 91770 utilities. Using these utilities, the job can be identified by prefixing the job number  
 91771 with '% '.
- 91772 <state> One of the following strings (in the POSIX locale):
- 91773 **Running** Indicates that the job has not been suspended by a signal and has not  
 91774 exited.
- 91775 **Done** Indicates that the job completed and returned exit status zero.
- 91776 **Done(code)** Indicates that the job completed normally and that it exited with the  
 91777 specified non-zero exit status, *code*, expressed as a decimal number.
- 91778 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.
- 91779 **Stopped (SIGTSTP)**  
 91780 Indicates that the job was suspended by the SIGTSTP signal.
- 91781 **Stopped (SIGSTOP)**  
 91782 Indicates that the job was suspended by the SIGSTOP signal.
- 91783 **Stopped (SIGTTIN)**  
 91784 Indicates that the job was suspended by the SIGTTIN signal.
- 91785 **Stopped (SIGTTOU)**  
 91786 Indicates that the job was suspended by the SIGTTOU signal.
- 91787 The implementation may substitute the string **Suspended** in place of **Stopped**. If  
 91788 the job was terminated by a signal, the format of <state> is unspecified, but it shall  
 91789 be visibly distinct from all of the other <state> formats shown here and shall  
 91790 indicate the name or description of the signal causing the termination.
- 91791 <command> The associated command that was given to the shell.
- 91792 If the **-l** option is specified, a field containing the process group ID shall be inserted before the  
 91793 <state> field. Also, more processes in a process group may be output on separate lines, using

91794 only the process ID and *<command>* fields.

#### 91795 **STDERR**

91796 The standard error shall be used only for diagnostic messages.

#### 91797 **OUTPUT FILES**

91798 None.

#### 91799 **EXTENDED DESCRIPTION**

91800 None.

#### 91801 **EXIT STATUS**

91802 The following exit values shall be returned:

91803 0 Successful completion.

91804 >0 An error occurred.

#### 91805 **CONSEQUENCES OF ERRORS**

91806 Default.

#### 91807 **APPLICATION USAGE**

91808 The *-p* option is the only portable way to find out the process group of a job because different  
91809 implementations have different strategies for defining the process group of the job. Usage such  
91810 as *\$(jobs -p)* provides a way of referring to the process group of the job in an implementation-  
91811 independent way.

91812 The *jobs* utility does not work as expected when it is operating in its own utility execution  
91813 environment because that environment has no applicable jobs to manipulate. See the  
91814 APPLICATION USAGE section for *bg*. For this reason, *jobs* is generally implemented as a shell  
91815 regular built-in.

#### 91816 **EXAMPLES**

91817 None.

#### 91818 **RATIONALE**

91819 Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity—the  
91820 "%%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical  
91821 practice of the KornShell and the C shell with job control.

91822 The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard  
91823 developers examined the characteristics of the C shell versions of these utilities and found that  
91824 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for  
91825 this volume of POSIX.1-2008 to maintain a degree of uniformity with the rest of the KornShell  
91826 features selected (such as the very popular command line editing features).

91827 The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*  
91828 utilities because *jobs* is useful for examining background jobs, regardless of the condition of job  
91829 control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*  
91830 can still be used to examine the background jobs associated with that current session. Similarly,  
91831 *kill* can then be used to kill background jobs with *kill %<background job number>*.

91832 The output for terminated jobs is left unspecified to accommodate various historical systems.  
91833 The following formats have been witnessed:

91834 1. **Killed**(*signal name*)

91835 2. *signal name*

- 91836 3. *signal name*(**coredump**)
- 91837 4. *signal description*– **core dumped**
- 91838 Most users should be able to understand these formats, although it means that applications have  
91839 trouble parsing them.
- 91840 The calculation of job IDs was not described since this would suggest an implementation, which  
91841 may impose unnecessary restrictions.
- 91842 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,  
91843 exited, or stopped since the last status report”. It was removed because the shell always writes  
91844 any changed status of jobs before each prompt.
- 91845 **FUTURE DIRECTIONS**
- 91846 None.
- 91847 **SEE ALSO**
- 91848 [Section 2.12](#) (on page 2331), *bg, fg, kill, wait*
- 91849 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 91850 **CHANGE HISTORY**
- 91851 First released in Issue 4.
- 91852 **Issue 6**
- 91853 This utility is marked as part of the User Portability Utilities option.
- 91854 The JC shading is removed as job control is mandatory in this version.
- 91855 **Issue 7**
- 91856 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91857 **NAME**

91858 join — relational database operator

91859 **SYNOPSIS**91860 join **[-a file\_number|-v file\_number] [-e string] [-o list] [-t char]**  
91861 **[-1 field] [-2 field] file1 file2**91862 **DESCRIPTION**91863 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be  
91864 written to the standard output.91865 The join field is a field in each file on which the files are compared. The *join* utility shall write  
91866 one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The  
91867 output line by default shall consist of the join field, then the remaining fields from *file1*, then the  
91868 remaining fields from *file2*. This format can be changed by using the **-o** option (see below). The  
91869 **-a** option can be used to add unmatched lines to the output. The **-v** option can be used to  
91870 output only unmatched lines.91871 The files *file1* and *file2* shall be ordered in the collating sequence of **sort -b** on the fields on which  
91872 they shall be joined, by default the first in each line. All selected output shall be written in the  
91873 same collating sequence.91874 The default input field separators shall be <blank> characters. In this case, multiple separators  
91875 shall count as one field separator, and leading separators shall be ignored. The default output  
91876 field separator shall be a <space>.91877 The field separator and collating sequence can be changed by using the **-t** option (see below).91878 If the same key appears more than once in either file, all combinations of the set of remaining  
91879 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines  
91880 encountered.

91881 If the input files are not in the appropriate collating sequence, the results are unspecified.

91882 **OPTIONS**91883 The *join* utility shall conform to XBD [Section 12.2](#) (on page 215).

91884 The following options shall be supported:

91885 **-a file\_number**91886 Produce a line for each unpairable line in file *file\_number*, where *file\_number* is 1 or  
91887 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable  
91888 lines shall be output.91889 **-e string** Replace empty output fields in the list selected by **-o** with the string *string*.91890 **-o list** Construct the output line to comprise the fields specified in *list*, each element of  
91891 which shall have one of the following two forms:

- 91892 1.
- file\_number.field*
- , where
- file\_number*
- is a file number and
- field*
- is a decimal
- 
- 91893 integer field number
- 
- 91894 2. 0 (zero), representing the join field

91895 The elements of *list* shall be either <comma>-separated or <blank>-separated, as  
91896 specified in Guideline 8 of XBD [Section 12.2](#) (on page 215). The fields specified by  
91897 *list* shall be written for all selected output lines. Fields selected by *list* that do not  
91898 appear in the input shall be treated as empty output fields. (See the **-e** option.)  
91899 Only specifically requested fields shall be written. The application shall ensure that  
91900 *list* is a single command line argument.

## join

Utilities

- 91901        **-t char**        Use character *char* as a separator, for both input and output. Every appearance of *char* in a line shall be significant. When this option is specified, the collating sequence shall be the same as *sort* without the **-b** option.
- 91902
- 91903
- 91904        **-v file\_number**
- 91905                    Instead of the default output, produce a line only for each unpairable line in *file\_number*, where *file\_number* is 1 or 2. If both **-v1** and **-v2** are specified, all unpairable lines shall be output.
- 91906
- 91907
- 91908        **-1 field**        Join on the *fieldth* field of file 1. Fields are decimal integers starting with 1.
- 91909        **-2 field**        Join on the *fieldth* field of file 2. Fields are decimal integers starting with 1.
- 91910        **OPERANDS**
- 91911                    The following operands shall be supported:
- 91912        *file1, file2*        A pathname of a file to be joined. If either of the *file1* or *file2* operands is *'-'*, the standard input shall be used in its place.
- 91913
- 91914        **STDIN**
- 91915                    The standard input shall be used only if the *file1* or *file2* operand is *'-'*. See the INPUT FILES section.
- 91916
- 91917        **INPUT FILES**
- 91918                    The input files shall be text files.
- 91919        **ENVIRONMENT VARIABLES**
- 91920                    The following environment variables shall affect the execution of *join*:
- 91921        *LANG*                Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
- 91922
- 91923
- 91924        *LC\_ALL*                If set to a non-empty string value, override the values of all the other internationalization variables.
- 91925
- 91926        *LC\_COLLATE*
- 91927                    Determine the locale of the collating sequence *join* expects to have been used when the input files were sorted.
- 91928
- 91929        *LC\_CTYPE*             Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
- 91930
- 91931
- 91932        *LC\_MESSAGES*
- 91933                    Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 91934
- 91935        XSI        *NLSPATH*             Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 91936        **ASYNCHRONOUS EVENTS**
- 91937                    Default.
- 91938        **STDOUT**
- 91939                    The *join* utility output shall be a concatenation of selected character fields. When the **-o** option is not specified, the output shall be:
- 91940
- 91941                    "%s%s%s\n", <join field>, <other file1 fields>,  
91942                    <other file2 fields>
- 91943                    If the join field is not the first field in a file, the <other file fields> for that file shall be:

91944 <fields preceding join field>, <fields following join field>

91945 When the **-o** option is specified, the output format shall be:

91946 "%s\n", <concatenation of fields>

91947 where the concatenation of fields is described by the **-o** option, above.

91948 For either format, each field (except the last) shall be written with its trailing separator character.  
91949 If the separator is the default (<blank> characters), a single <space> shall be written after each  
91950 field (except the last).

#### 91951 **STDERR**

91952 The standard error shall be used only for diagnostic messages.

#### 91953 **OUTPUT FILES**

91954 None.

#### 91955 **EXTENDED DESCRIPTION**

91956 None.

#### 91957 **EXIT STATUS**

91958 The following exit values shall be returned:

91959 0 All input files were output successfully.

91960 >0 An error occurred.

#### 91961 **CONSEQUENCES OF ERRORS**

91962 Default.

#### 91963 **APPLICATION USAGE**

91964 Pathnames consisting of numeric digits or of the form *string.string* should not be specified  
91965 directly following the **-o** list.

#### 91966 **EXAMPLES**

91967 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

| 91968 | !Name   | Phone Number    |
|-------|---------|-----------------|
| 91969 | Don     | +1 123-456-7890 |
| 91970 | Hal     | +1 234-567-8901 |
| 91971 | Yasushi | +2 345-678-9012 |

91972 and file **fax**:

| 91973 | !Name   | Fax Number      |
|-------|---------|-----------------|
| 91974 | Don     | +1 123-456-7899 |
| 91975 | Keith   | +1 456-789-0122 |
| 91976 | Yasushi | +2 345-678-9011 |

91977 (where the large expanses of white space are meant to each represent a single <tab>), the  
91978 command:

91979 `join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax`

91980 would produce:

| 91981 | !Name   | Phone Number    | Fax Number      |
|-------|---------|-----------------|-----------------|
| 91982 | Don     | +1 123-456-7890 | +1 123-456-7899 |
| 91983 | Hal     | +1 234-567-8901 | (unknown)       |
| 91984 | Keith   | (unknown)       | +1 456-789-0122 |
| 91985 | Yasushi | +2 345-678-9012 | +2 345-678-9011 |

91986 Multiple instances of the same key will produce combinatorial results. The following:

91987 fa:

91988 a x

91989 a y

91990 a z

91991 fb:

91992 a p

91993 will produce:

91994 a x p

91995 a y p

91996 a z p

91997 And the following:

91998 fa:

91999 a b c

92000 a d e

92001 fb:

92002 a w x

92003 a y z

92004 a o p

92005 will produce:

92006 a b c w x

92007 a b c y z

92008 a b c o p

92009 a d e w x

92010 a d e y z

92011 a d e o p

## 92012 RATIONALE

92013 The `-e` option is only effective when used with `-o` because, unless specific fields are identified  
 92014 using `-o`, `join` is not aware of what fields might be empty. The exception to this is the join field,  
 92015 but identifying an empty join field with the `-e` string is not historical practice and some scripts  
 92016 might break if this were changed.

92017 The 0 field in the `-o` list was adopted from the Tenth Edition version of `join` to satisfy  
 92018 international objections that the `join` in the base documents does not support the “full join” or  
 92019 “outer join” described in relational database literature. Although it has been possible to include  
 92020 a join field in the output (by default, or by field number using `-o`), the join field could not be  
 92021 included for an unpaired line selected by `-a`. The `-o 0` field essentially selects the union of the  
 92022 join fields.

92023 This sort of outer join was not possible with the `join` commands in the base documents. The `-o 0`  
 92024 field was chosen because it is an upwards-compatible change for applications. An alternative  
 92025 was considered: have the join field represent the union of the fields in the files (where they are  
 92026 identical for matched lines, and one or both are null for unmatched lines). This was not adopted  
 92027 because it would break some historical applications.

92028 The ability to specify `file2` as `-` is not historical practice; it was added for completeness.

92029 The `-v` option is not historical practice, but was considered necessary because it permitted the  
 92030 writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which  
 92031 prints both lines that do and do not match. This additional facility is parallel with the `-v` option

- 92032 of *grep*.
- 92033 Some historical implementations have been encountered where a blank line in one of the input  
92034 files was considered to be the end of the file; the description in this volume of POSIX.1-2008 does  
92035 not cite this as an allowable case.
- 92036 Earlier versions of this standard allowed *-j*, *-j1*, *-j2* options, and a form of the *-o* option that  
92037 allowed the *list* option-argument to be multiple arguments. These forms are no longer specified  
92038 by POSIX.1-2008 but may be present in some implementations.
- 92039 **FUTURE DIRECTIONS**
- 92040 None.
- 92041 **SEE ALSO**
- 92042 *awk*, *comm*, *sort*, *uniq*
- 92043 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
- 92044 **CHANGE HISTORY**
- 92045 First released in Issue 2.
- 92046 **Issue 6**
- 92047 The obsolescent *-j* options and the multi-argument *-o* option are removed in this version.
- 92048 The normative text is reworded to avoid use of the term “must” for application requirements.
- 92049 **Issue 7**
- 92050 Austin Group Interpretation 1003.1-2001 #027 is applied.
- 92051 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**kill**

Utilities

92052 **NAME**92053 `kill` — terminate or signal processes92054 **SYNOPSIS**92055 `kill -s signal_name pid...`92056 `kill -l [exit_status]`92057 XSI `kill [-signal_name] pid...`92058 `kill [-signal_number] pid...`92059 **DESCRIPTION**92060 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.92061 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function  
92062 defined in the System Interfaces volume of POSIX.1-2008 called with the following arguments:

- 92063 • The value of the *pid* operand shall be used as the *pid* argument.
- 92064 • The *sig* argument is the value specified by the `-s` option, `-signal_number` option, or the  
92065 `-signal_name` option, or by SIGTERM, if none of these options is specified.

92066 **OPTIONS**92067 XSI The *kill* utility shall conform to XBD Section 12.2 (on page 215), except that in the last two  
92068 SYNOPSIS forms, the `-signal_number` and `-signal_name` options are usually more than a single  
92069 character.

92070 The following options shall be supported:

92071 `-l` (The letter ell.) Write all values of *signal\_name* supported by the implementation, if  
92072 no operand is given. If an *exit\_status* operand is given and it is a value of the `'?'`  
92073 shell special parameter (see Section 2.5.2 (on page 2302) and *wait*) corresponding to  
92074 a process that was terminated by a signal, the *signal\_name* corresponding to the  
92075 signal that terminated the process shall be written. If an *exit\_status* operand is  
92076 given and it is the unsigned decimal integer value of a signal number, the  
92077 *signal\_name* (the symbolic constant name without the **SIG** prefix defined in the  
92078 Base Definitions volume of POSIX.1-2008) corresponding to that signal shall be  
92079 written. Otherwise, the results are unspecified.

92080 `-s signal_name`

92081 Specify the signal to send, using one of the symbolic names defined in the  
92082 `<signal.h>` header. Values of *signal\_name* shall be recognized in a case-independent  
92083 fashion, without the **SIG** prefix. In addition, the symbolic name `0` shall be  
92084 recognized, representing the signal value zero. The corresponding signal shall be  
92085 sent instead of SIGTERM.

92086 XSI `-signal_name`92087 Equivalent to `-s signal_name`.92088 XSI `-signal_number`

92089 Specify a non-negative decimal integer, *signal\_number*, representing the signal to be  
92090 used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The  
92091 correspondence between integer values and the *sig* value used is shown in the  
92092 following list.

92093 The effects of specifying any *signal\_number* other than those listed below are  
92094 undefined.

|       |                                                                                                       |         |
|-------|-------------------------------------------------------------------------------------------------------|---------|
| 92095 | 0                                                                                                     | 0       |
| 92096 | 1                                                                                                     | SIGHUP  |
| 92097 | 2                                                                                                     | SIGINT  |
| 92098 | 3                                                                                                     | SIGQUIT |
| 92099 | 6                                                                                                     | SIGABRT |
| 92100 | 9                                                                                                     | SIGKILL |
| 92101 | 14                                                                                                    | SIGALRM |
| 92102 | 15                                                                                                    | SIGTERM |
| 92103 | If the first argument is a negative integer, it shall be interpreted as a <code>-signal_number</code> |         |
| 92104 | option, not as a negative <code>pid</code> operand specifying a process group.                        |         |

**OPERANDS**

92106 The following operands shall be supported:

92107 `pid` One of the following:

- 92108 1. A decimal integer specifying a process or process group to be signaled. The  
92109 process or processes selected by positive, negative, and zero values of the  
92110 `pid` operand shall be as described for the `kill()` function. If process number 0  
92111 is specified, all processes in the current process group shall be signaled. For  
92112 the effects of negative `pid` numbers, see the `kill()` function defined in the  
92113 System Interfaces volume of POSIX.1-2008. If the first `pid` operand is  
92114 negative, it should be preceded by "--" to keep it from being interpreted as  
92115 an option.
- 92116 2. A job control job ID (see XBD Section 3.203, on page 65) that identifies a  
92117 background process group to be signaled. The job control job ID notation is  
92118 applicable only for invocations of `kill` in the current shell execution  
92119 environment; see Section 2.12 (on page 2331).

92120 `exit_status` A decimal integer specifying a signal number or the exit status of a process  
92121 terminated by a signal.

**STDIN**

92123 Not used.

**INPUT FILES**

92125 None.

**ENVIRONMENT VARIABLES**

92127 The following environment variables shall affect the execution of `kill`:

- 92128 `LANG` Provide a default value for the internationalization variables that are unset or null.  
92129 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
92130 variables used to determine the values of locale categories.)
- 92131 `LC_ALL` If set to a non-empty string value, override the values of all the other  
92132 internationalization variables.
- 92133 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
92134 characters (for example, single-byte as opposed to multi-byte characters in  
92135 arguments).

**kill**

Utilities

- 92136 *LC\_MESSAGES*
- 92137 Determine the locale that should be used to affect the format and contents of
- 92138 diagnostic messages written to standard error.
- 92139 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 92140 **ASYNCHRONOUS EVENTS**
- 92141 Default.
- 92142 **STDOUT**
- 92143 When the `-l` option is not specified, the standard output shall not be used.
- 92144 When the `-l` option is specified, the symbolic name of each signal shall be written in the
- 92145 following format:
- 92146 `"%s%c", <signal_name>, <separator>`
- 92147 where the `<signal_name>` is in uppercase, without the **SIG** prefix, and the `<separator>` shall be
- 92148 either a `<newline>` or a `<space>`. For the last signal written, `<separator>` shall be a `<newline>`.
- 92149 When both the `-l` option and `exit_status` operand are specified, the symbolic name of the
- 92150 corresponding signal shall be written in the following format:
- 92151 `"%s\n", <signal_name>`
- 92152 **STDERR**
- 92153 The standard error shall be used only for diagnostic messages.
- 92154 **OUTPUT FILES**
- 92155 None.
- 92156 **EXTENDED DESCRIPTION**
- 92157 None.
- 92158 **EXIT STATUS**
- 92159 The following exit values shall be returned:
- 92160 0 At least one matching process was found for each `pid` operand, and the specified signal was
- 92161 successfully processed for at least one matching process.
- 92162 >0 An error occurred.
- 92163 **CONSEQUENCES OF ERRORS**
- 92164 Default.
- 92165 **APPLICATION USAGE**
- 92166 Process numbers can be found by using `ps`.
- 92167 The job control job ID notation is not required to work as expected when `kill` is operating in its
- 92168 own utility execution environment. In either of the following examples:
- 92169 `nohup kill %1 &`
- 92170 `system("kill %1");`
- 92171 the `kill` operates in a different environment and does not share the shell's understanding of job
- 92172 numbers.
- 92173 **EXAMPLES**
- 92174 Any of the commands:
- 92175 `kill -9 100 -165`
- 92176 `kill -s kill 100 -165`

92177 `kill -s KILL 100 -165`

92178 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose  
92179 process group ID is 165, assuming the sending process has permission to send that signal to the  
92180 specified processes, and that they exist.

92181 The System Interfaces volume of POSIX.1-2008 and this volume of POSIX.1-2008 do not require  
92182 specific signal numbers for any *signal\_names*. Even the *-signal\_number* option provides symbolic  
92183 (although numeric) names for signals. If a process is terminated by a signal, its exit status  
92184 indicates the signal that killed it, but the exact values are not specified. The *kill -l* option,  
92185 however, can be used to map decimal signal numbers and exit status values into the name of a  
92186 signal. The following example reports the status of a terminated job:

```
92187 job
92188 stat=$?
92189 if [ $stat -eq 0 ]
92190 then
92191     echo job completed successfully.
92192 elif [ $stat -gt 128 ]
92193 then
92194     echo job terminated by signal SIG$(kill -l $stat).
92195 else
92196     echo job terminated with error code $stat.
92197 fi
```

92198 To send the default signal to a process group (say 123), an application should use a command  
92199 similar to one of the following:

```
92200 kill -TERM -123
92201 kill -- -123
```

#### 92202 RATIONALE

92203 The *-l* option originated from the C shell, and is also implemented in the KornShell. The C shell  
92204 output can consist of multiple output lines because the signal names do not always fit on a  
92205 single line on some terminal screens. The KornShell output also included the implementation-  
92206 defined signal numbers and was considered by the standard developers to be too difficult for  
92207 scripts to parse conveniently. The specified output format is intended not only to accommodate  
92208 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing  
92209 on systems for which this is appropriate.

92210 An early proposal invented the name SIGNULL as a *signal\_name* for signal 0 (used by the System  
92211 Interfaces volume of POSIX.1-2008 to test for the existence of a process without sending it a  
92212 signal). Since the *signal\_name* 0 can be used in this case unambiguously, SIGNULL has been  
92213 removed.

92214 An early proposal also required symbolic *signal\_names* to be recognized with or without the **SIG**  
92215 prefix. Historical versions of *kill* have not written the **SIG** prefix for the *-l* option and have not  
92216 recognized the **SIG** prefix on *signal\_names*. Since neither applications portability nor ease-of-use  
92217 would be improved by requiring this extension, it is no longer required.

92218 To avoid an ambiguity of an initial negative number argument specifying either a signal number  
92219 or a process group, POSIX.1-2008 mandates that it is always considered the former by  
92220 implementations that support the XSI option. It also requires that conforming applications  
92221 always use the "--" options terminator argument when specifying a process group, unless an  
92222 option is also specified.

92223 The *-s* option was added in response to international interest in providing some form of *kill* that

**kill**

Utilities

- 92224 meets the Utility Syntax Guidelines.
- 92225 The job control job ID notation is not required to work as expected when *kill* is operating in its  
 92226 own utility execution environment. In either of the following examples:
- 92227 `nohup kill %1 &`  
 92228 `system("kill %1");`
- 92229 the *kill* operates in a different environment and does not understand how the shell has managed  
 92230 its job numbers.
- 92231 **FUTURE DIRECTIONS**  
 92232 None.
- 92233 **SEE ALSO**  
 92234 [Chapter 2](#) (on page 2297), *ps*, *wait*
- 92235 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215), [<signal.h>](#)  
 92236 XSH *kill()*
- 92237 **CHANGE HISTORY**  
 92238 First released in Issue 2.
- 92239 **Issue 6**  
 92240 The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI  
 92241 option, corresponding to a similar change in the *trap* special built-in.
- 92242 **Issue 7**  
 92243 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92244 **NAME**92245 lex — generate programs for lexical tasks (**DEVELOPMENT**)92246 **SYNOPSIS**92247 CD `lex [-t] [-n|-v] [file...]`92248 **DESCRIPTION**

92249 The *lex* utility shall generate C programs to be used in lexical processing of character input, and  
 92250 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code  
 92251 and conform to the ISO C standard, without depending on any undefined, unspecified, or  
 92252 implementation-defined behavior, except in cases where the code is copied directly from the  
 92253 supplied source, or in cases that are documented by the implementation. Usually, the *lex* utility  
 92254 shall write the program it generates to the file **lex.yy.c**; the state of this file is unspecified if *lex*  
 92255 exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete  
 92256 description of the *lex* input language.

92257 **OPTIONS**92258 The *lex* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

92259 The following options shall be supported:

- 92260 **-n** Suppress the summary of statistics usually written with the **-v** option. If no table  
 92261 sizes are specified in the *lex* source code and the **-v** option is not specified, then **-n**  
 92262 is implied.
- 92263 **-t** Write the resulting program to standard output instead of **lex.yy.c**.
- 92264 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*  
 92265 table sizes in Definitions in *lex* (on page 2828).) If the **-t** option is specified and **-n**  
 92266 is not specified, this report shall be written to standard error. If table sizes are  
 92267 specified in the *lex* source code, and if the **-n** option is not specified, the **-v** option  
 92268 may be enabled.

92269 **OPERANDS**

92270 The following operand shall be supported:

- 92271 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be  
 92272 concatenated to produce a single *lex* program. If no *file* operands are specified, or if  
 92273 a *file* operand is **'-'**, the standard input shall be used.

92274 **STDIN**

92275 The standard input shall be used if no *file* operands are specified, or if a *file* operand is **'-'**. See  
 92276 INPUT FILES.

92277 **INPUT FILES**

92278 The input files shall be text files containing *lex* source code, as described in the EXTENDED  
 92279 DESCRIPTION section.

92280 **ENVIRONMENT VARIABLES**92281 The following environment variables shall affect the execution of *lex*:

- 92282 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 92283 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 92284 variables used to determine the values of locale categories.)
- 92285 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 92286 internationalization variables.

- 92287 *LC\_COLLATE*
- 92288 Determine the locale for the behavior of ranges, equivalence classes, and multi-
- 92289 character collating elements within regular expressions. If this variable is not set to
- 92290 the POSIX locale, the results are unspecified.
- 92291 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 92292 characters (for example, single-byte as opposed to multi-byte characters in
- 92293 arguments and input files), and the behavior of character classes within regular
- 92294 expressions. If this variable is not set to the POSIX locale, the results are
- 92295 unspecified.
- 92296 *LC\_MESSAGES*
- 92297 Determine the locale that should be used to affect the format and contents of
- 92298 diagnostic messages written to standard error.
- 92299 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 92300 **ASYNCHRONOUS EVENTS**
- 92301 Default.
- 92302 **STDOUT**
- 92303 If the *-t* option is specified, the text file of C source code output of *lex* shall be written to
- 92304 standard output.
- 92305 If the *-t* option is not specified:
- 92306 • Implementation-defined informational, error, and warning messages concerning the
  - 92307 contents of *lex* source code input shall be written to either the standard output or standard
  - 92308 error.
  - 92309 • If the *-v* option is specified and the *-n* option is not specified, *lex* statistics shall also be
  - 92310 written to either the standard output or standard error, in an implementation-defined
  - 92311 format. These statistics may also be generated if table sizes are specified with a '*%*'
  - 92312 operator in the *Definitions* section, as long as the *-n* option is not specified.
- 92313 **STDERR**
- 92314 If the *-t* option is specified, implementation-defined informational, error, and warning messages
- 92315 concerning the contents of *lex* source code input shall be written to the standard error.
- 92316 If the *-t* option is not specified:
- 92317 1. Implementation-defined informational, error, and warning messages concerning the
  - 92318 contents of *lex* source code input shall be written to either the standard output or
  - 92319 standard error.
  - 92320 2. If the *-v* option is specified and the *-n* option is not specified, *lex* statistics shall also be
  - 92321 written to either the standard output or standard error, in an implementation-defined
  - 92322 format. These statistics may also be generated if table sizes are specified with a '*%*'
  - 92323 operator in the *Definitions* section, as long as the *-n* option is not specified.
- 92324 **OUTPUT FILES**
- 92325 A text file containing C source code shall be written to *lex.yy.c*, or to the standard output if the *-t*
- 92326 option is present.
- 92327 **EXTENDED DESCRIPTION**
- 92328 Each input file shall contain *lex* source code, which is a table of regular expressions with
- 92329 corresponding actions in the form of C program fragments.
- 92330 When *lex.yy.c* is compiled and linked with the *lex* library (using the *-ll* operand with *c99*), the

92331 resulting program shall read character input from the standard input and shall partition it into  
92332 strings that match the given expressions.

92333 When an expression is matched, these actions shall occur:

- 92334 • The input string that was matched shall be left in *yytext* as a null-terminated string; *yytext*  
92335 shall either be an external character array or a pointer to a character string. As explained in  
92336 [Definitions in lex](#) (on page 2828), the type can be explicitly selected using the `%array` or  
92337 `%pointer` declarations, but the default is implementation-defined.
- 92338 • The external `int yyleng` shall be set to the length of the matching string.
- 92339 • The expression's corresponding program fragment, or action, shall be executed.

92340 During pattern matching, *lex* shall search the set of patterns for the single longest possible  
92341 match. Among rules that match the same number of characters, the rule given first shall be  
92342 chosen.

92343 The general format of *lex* source shall be:

```
92344     Definitions
92345     %%
92346     Rules
92347     %%
92348     UserSubroutines
```

92349 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);  
92350 the second "%%" is required only if user subroutines follow.

92351 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program  
92352 fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly,  
92353 anything in the *Definitions* section included between delimiter lines containing only "%{" and  
92354 "%}" shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

92355 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing  
92356 at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c`  
92357 after the declarations of variables for the `yylex()` function and before the first line of code in  
92358 `yylex()`. Thus, user variables local to `yylex()` can be declared here, as well as application code to  
92359 execute upon entry to `yylex()`.

92360 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"  
92361 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is  
92362 undefined. The presence of such input may result in an erroneous definition of the `yylex()`  
92363 function.

92364 C-language code in the input shall not contain C-language trigraphs. The C-language code  
92365 within "%{" and "%}" delimiter lines shall not contain any lines consisting only of "%}", or  
92366 only of "%%".

92367 **Definitions in lex**

92368 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between  
 92369 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*  
 92370 substitution string. The format of these lines shall be:

92371 *name substitute*

92372 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is  
 92373 undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name*  
 92374 string shall be recognized in this context only when the braces are provided and when it does  
 92375 not appear within a bracket expression or within double-quotes.

92376 In the *Definitions* section, any line beginning with a <percent-sign> ('%') character and followed  
 92377 by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.  
 92378 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall  
 92379 define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns  
 92380 with no state specified shall be also active; in a %x state, such patterns shall not be active. The  
 92381 rest of the line, after the first word, shall be considered to be one or more <blank>-separated  
 92382 names of start conditions. Start condition names shall be constructed in the same way as  
 92383 definition names. Start conditions can be used to restrict the matching of regular expressions to  
 92384 one or more states as described in [Regular Expressions in lex](#) (on page 2829).

92385 Implementations shall accept either of the following two mutually-exclusive declarations in the  
 92386 *Definitions* section:

92387 **%array** Declare the type of *yytext* to be a null-terminated character array.

92388 **%pointer** Declare the type of *yytext* to be a pointer to a null-terminated character string.

92389 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of  
 92390 the scanner source file (that is, via an **extern**), the application shall include the appropriate  
 92391 **%array** or **%pointer** declaration in the scanner source file.

92392 Implementations shall accept declarations in the *Definitions* section for setting certain internal  
 92393 table sizes. The declarations are shown in the following table.

92394 **Table 4-10** Table Size Declarations in *lex*

| Declaration | Description                        | Minimum Value |
|-------------|------------------------------------|---------------|
| %p <i>n</i> | Number of positions                | 2500          |
| %n <i>n</i> | Number of states                   | 500           |
| %a <i>n</i> | Number of transitions              | 2000          |
| %e <i>n</i> | Number of parse tree nodes         | 1000          |
| %k <i>n</i> | Number of packed character classes | 1000          |
| %o <i>n</i> | Size of the output array           | 3000          |

92402 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>  
 92403 characters. The exact meaning of these table size numbers is implementation-defined. The  
 92404 implementation shall document how these numbers affect the *lex* utility and how they are  
 92405 related to any output that may be generated by the implementation should limitations be  
 92406 encountered during the execution of *lex*. It shall be possible to determine from this output  
 92407 which of the table size values needs to be modified to permit *lex* to successfully generate tables  
 92408 for the input language. The values in the column Minimum Value represent the lowest values  
 92409 conforming implementations shall provide.

92410 **Rules in lex**

92411 The rules in *lex* source files are a table in which the left column contains regular expressions and  
 92412 the right column contains actions (C program fragments) to be executed when the expressions  
 92413 are recognized.

92414 *ERE action*

92415 *ERE action*

92416 . . .

92417 The extended regular expression (ERE) portion of a row shall be separated from *action* by one or  
 92418 more <blank> characters. A regular expression containing <blank> characters shall be  
 92419 recognized under one of the following conditions:

- 92420 • The entire expression appears within double-quotes.
- 92421 • The <blank> characters appear within double-quotes or square brackets.
- 92422 • Each <blank> is preceded by a <backslash> character.

92423 **User Subroutines in lex**

92424 Anything in the user subroutines section shall be copied to *lex.yy.c* following *yylex()*.

92425 **Regular Expressions in lex**

92426 The *lex* utility shall support the set of extended regular expressions (see XBD Section 9.4, on page  
 92427 188), with the following additions and exceptions to the syntax:

92428 ". . ." Any string enclosed in double-quotes shall represent the characters within the  
 92429 double-quotes as themselves, except that <backslash>-escapes (which appear in  
 92430 the following table) shall be recognized. Any <backslash>-escape sequence shall be  
 92431 terminated by the closing quote. For example, "\01"1" represents a single  
 92432 string: the octal value 1 followed by the character ' 1 '.

92433 <state>*r*, <state1, state2, . . .>*r*

92434 The regular expression *r* shall be matched only when the program is in one of the  
 92435 start conditions indicated by *state*, *state1*, and so on; see [Actions in lex](#) (on page  
 92436 2831). (As an exception to the typographical conventions of the rest of this volume  
 92437 of POSIX.1-2008, in this case <state> does not represent a metavariable, but the  
 92438 literal angle-bracket characters surrounding a symbol.) The start condition shall be  
 92439 recognized as such only at the beginning of a regular expression.

92440 *r/x*

92441 The regular expression *r* shall be matched only if it is followed by an occurrence of  
 92442 regular expression *x* (*x* is the instance of trailing context, further defined below).  
 92443 The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches  
 92444 the beginning of *x*, the result is unspecified. The *r* expression cannot include  
 92445 further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include  
 92446 the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$'  
 92447 operator. That is, only one occurrence of trailing context is allowed in a *lex* regular  
 92448 expression, and the '^' operator only can be used at the beginning of such an  
 expression.

92449 {*name*}

92450 When *name* is one of the substitution symbols from the *Definitions* section, the  
 92451 string, including the enclosing braces, shall be replaced by the *substitute* value. The  
 92452 *substitute* value shall be treated in the extended regular expression as if it were  
 92453 enclosed in parentheses. No substitution shall occur if {*name*} occurs within a  
 bracket expression or within double-quotes.

92454 Within an ERE, a <backslash> character shall be considered to begin an escape sequence as  
 92455 specified in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r',  
 92456 '\t', '\v'). In addition, the escape sequences in the following table shall be recognized.

92457 A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to  
 92458 represent a <newline>. A <newline> shall not be matched by a period operator.

Table 4-11 Escape Sequences in *lex*

| Escape Sequence       | Description                                                                                                                                                                                                             | Meaning                                                                                                                                                                                                                         |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\digits</code>  | A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined. | The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte. |
| <code>\xdigits</code> | A <backslash> character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.  | The character whose encoding is represented by the hexadecimal integer.                                                                                                                                                         |
| <code>\c</code>       | A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').                                          | The character 'c', unchanged.                                                                                                                                                                                                   |

92481 **Note:** If a '\x' sequence needs to be immediately followed by a hexadecimal digit character, a  
 92482 sequence such as "\x1"1" can be used, which represents a character containing the value 1,  
 92483 followed by the character '1'.

92484 The order of precedence given to extended regular expressions for *lex* differs from that specified  
 92485 in XBD Section 94 (on page 188). The order of precedence for *lex* shall be as shown in the  
 92486 following table, from high to low.

92487 **Note:** The escaped characters entry is not meant to imply that these are operators, but they are  
 92488 included in the table to show their relationships to the true operators. The start condition,  
 92489 trailing context, and anchoring notations have been omitted from the table because of the  
 92490 placement restrictions described in this section; they can only appear at the beginning or ending  
 92491 of an ERE.

Table 4-12 ERE Precedence in *lex*

| Extended Regular Expression              | Precedence                    |
|------------------------------------------|-------------------------------|
| <i>collation-related bracket symbols</i> | [ = ] [ : : ] [ . . ]         |
| <i>escaped characters</i>                | \< <i>special character</i> > |
| <i>bracket expression</i>                | [ ]                           |
| <i>quoting</i>                           | " . . . "                     |
| <i>grouping</i>                          | ( )                           |
| <i>definition</i>                        | { <i>name</i> }               |
| <i>single-character RE duplication</i>   | * + ?                         |
| <i>concatenation</i>                     |                               |
| <i>interval expression</i>               | { <i>m</i> , <i>n</i> }       |
| <i>alternation</i>                       |                               |

The ERE anchoring operators '*^*' and '*\$*' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '*^*' operator can only be used at the beginning of an entire regular expression, and the '*\$*' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "*(^abc) | (def\$)*" is undefined; it can instead be written as two separate rules, one with the regular expression "*^abc*" and one with "*def\$*", which share a common action via the special '*|*' action (see below). If the pattern were written "*^abc|def\$*", it would match either "*abc*" or "*def*" on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "*(^|)foo(|\$)*" to match "*foo*" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

```

^foo/[ \n]      |
" foo"/[ \n]    /* Found foo as a separate word. */

```

Note also that '*\$*' is a form of trailing context (it is equivalent to "*/\n*") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator '*/'*' can be used as an ordinary character if presented within double-quotes, "*/"*"; preceded by a <backslash>, "*\"/*"; or within a bracket expression, "*[/]*". The start-condition '*<*' and '*>*' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

### Actions in *lex*

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement '*;*' shall be a valid action; any string in the *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```

ERE <one or more blanks> { program statement
                           program statement }

```

- 92537 The program statements shall not contain unbalanced curly brace preprocessing tokens.
- 92538 The default action when a string in the input to a `lex.yy.c` program is not matched by any  
92539 expression shall be to copy the string to the output. Because the default behavior of a program  
92540 generated by `lex` is to read the input and copy it to the output, a minimal `lex` source program that  
92541 has just `"%%"` shall generate a C program that simply copies the input to the output unchanged.
- 92542 Four special actions shall be available:
- 92543 | `ECHO;` | `REJECT;` | `BEGIN`
- 92544 | The action `'|'` means that the action for the next rule is the action for this rule.  
92545 Unlike the other three actions, `'|'` cannot be enclosed in braces or be  
92546 `<semicolon>`-terminated; the application shall ensure that it is specified alone, with  
92547 no other actions.
- 92548 **ECHO;** Write the contents of the string `yytext` on the output.
- 92549 **REJECT;** Usually only a single expression is matched by a given string in the input.  
92550 **REJECT** means "continue to the next expression that matches the current input",  
92551 and shall cause whatever rule was the second choice after the current rule to be  
92552 executed for the same input. Thus, multiple rules can be matched and executed for  
92553 one input string or overlapping input strings. For example, given the regular  
92554 expressions `"xyz"` and `"xy"` and the input `"xyz"`, usually only the regular  
92555 expression `"xyz"` would match. The next attempted match would start after `z`. If  
92556 the last action in the `"xyz"` rule is **REJECT**, both this rule and the `"xy"` rule  
92557 would be executed. The **REJECT** action may be implemented in such a fashion that  
92558 flow of control does not continue after it, as if it were equivalent to a **goto**  
92559 to another part of `yylex()`. The use of **REJECT** may result in somewhat larger and  
92560 slower scanners.
- 92561 **BEGIN** The action:
- 92562 `BEGIN newstate;`
- 92563 switches the state (start condition) to `newstate`. If the string `newstate` has not been  
92564 declared previously as a start condition in the *Definitions* section, the results are  
92565 unspecified. The initial state is indicated by the digit `'0'` or the token **INITIAL**.
- 92566 The functions or macros described below are accessible to user code included in the `lex` input. It  
92567 is unspecified whether they appear in the C code output of `lex`, or are accessible only through the  
92568 `-ll` operand to `c99` (the `lex` library).
- 92569 **int yylex(void)**  
92570 Performs lexical analysis on the input; this is the primary function generated by the `lex`  
92571 utility. The function shall return zero when the end of input is reached; otherwise, it shall  
92572 return non-zero values (tokens) determined by the actions that are selected.
- 92573 **int yymore(void)**  
92574 When called, indicates that when the next input string is recognized, it is to be appended to  
92575 the current value of `yytext` rather than replacing it; the value in `yylen` shall be adjusted  
92576 accordingly.
- 92577 **int yyless(int n)**  
92578 Retains `n` initial characters in `yytext`, NUL-terminated, and treats the remaining characters as  
92579 if they had not been read; the value in `yylen` shall be adjusted accordingly.

- 92580 **int input(void)**  
 92581 Returns the next character from the input, or zero on end-of-file. It shall obtain input from  
 92582 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning  
 92583 has begun, the effect of altering the value of *yyin* is undefined. The character read shall be  
 92584 removed from the input stream of the scanner without any processing by the scanner.
- 92585 **int unput(int c)**  
 92586 Returns the character 'c' to the input; *yytext* and *yytext* are undefined until the next  
 92587 expression is matched. The result of using *unput()* for more characters than have been input  
 92588 is unspecified.
- 92589 The following functions shall appear only in the *lex* library accessible through the `-D` operand;  
 92590 they can therefore be redefined by a conforming application:
- 92591 **int yywrap(void)**  
 92592 Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application  
 92593 requires *yylex()* to continue processing with another source of input, then the application  
 92594 can include a function *yywrap()*, which associates another file with the external variable  
 92595 `FILE * yyin` and shall return a value of zero.
- 92596 **int main(int argc, char \*argv[ ])**  
 92597 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to  
 92598 perform application-specific operations, calling *yylex()* as applicable.
- 92599 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin  
 92600 with the prefix `yy` or `YY`.
- 92601 **EXIT STATUS**  
 92602 The following exit values shall be returned:  
 92603 0 Successful completion.  
 92604 >0 An error occurred.
- 92605 **CONSEQUENCES OF ERRORS**  
 92606 Default.
- 92607 **APPLICATION USAGE**  
 92608 Conforming applications are warned that in the *Rules* section, an ERE without an action is not  
 92609 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or  
 92610 runtime errors.
- 92611 The purpose of *input()* is to take characters off the input stream and discard them as far as the  
 92612 lexical analysis is concerned. A common use is to discard the body of a comment once the  
 92613 beginning of a comment is recognized.
- 92614 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*  
 92615 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer  
 92616 interpret the regular expressions given in the *lex* source according to the environment specified  
 92617 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.  
 92618 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the  
 92619 lexical requirements of the input language being described, which is frequently locale-specific  
 92620 anyway. (For example, writing an analyzer that is used for French text is not automatically  
 92621 useful for processing other languages.)

92622 **EXAMPLES**

92623 The following is an example of a *lex* program that implements a rudimentary scanner for a  
92624 Pascal-like syntax:

```

92625     %{
92626     /* Need this for the call to atof() below. */
92627     #include <math.h>
92628     /* Need this for printf(), fopen(), and stdin below. */
92629     #include <stdio.h>
92630     %}

92631     DIGIT      [0-9]
92632     ID         [a-z][a-z0-9]*

92633     %%

92634     {DIGIT}+ {
92635         printf("An integer: %s (%d)\n", yytext,
92636             atoi(yytext));
92637     }

92638     {DIGIT}+"."{DIGIT}* {
92639         printf("A float: %s (%g)\n", yytext,
92640             atof(yytext));
92641     }

92642     if|then|begin|end|procedure|function {
92643         printf("A keyword: %s\n", yytext);
92644     }

92645     {ID}      printf("An identifier: %s\n", yytext);
92646     "+"|"-"|"*"|"|"      printf("An operator: %s\n", yytext);
92647     "{"[^]\n}*"      /* Eat up one-line comments. */
92648     [ \t\n]+      /* Eat up white space. */
92649     .      printf("Unrecognized character: %s\n", yytext);
92650     %%

92651     int main(int argc, char *argv[])
92652     {
92653         ++argv, --argc; /* Skip over program name. */
92654         if (argc > 0)
92655             yyin = fopen(argv[0], "r");
92656         else
92657             yyin = stdin;
92658         yylex();
92659     }

```

92660 **RATIONALE**

92661 Even though the `-c` option and references to the C language are retained in this description, *lex*  
92662 may be generalized to other languages, as was done at one time for EFL, the Extended  
92663 FORTRAN Language. Since the *lex* input specification is essentially language-independent,  
92664 versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are  
92665 known historical implementations that do so.

- 92666 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*  
 92667 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is  
 92668 assumed to be presented in the POSIX locale, but input and output are in the locale specified by  
 92669 the environment variables), then the tables in the lexical analyzer produced by *lex* would  
 92670 interpret EREs specified in the *lex* source in terms of the environment variables specified when  
 92671 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs  
 92672 given in the *lex* source according to the environment specified when the lexical analyzer is  
 92673 executed, but this is not possible with the current *lex* technology.
- 92674 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard  
 92675 use of escape sequences.
- 92676 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
 92677 but this has been modified in this version.
- 92678 There is no detailed output format specification. The observed behavior of *lex* under four  
 92679 different historical implementations was that none of these implementations consistently  
 92680 reported the line numbers for error and warning messages. Furthermore, there was a desire that  
 92681 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified  
 92682 avoids these formatting questions and problems with internationalization.
- 92683 Although the %x specifier for *exclusive* start conditions is not historical practice, it is believed to  
 92684 be a minor change to historical implementations and greatly enhances the usability of *lex*  
 92685 programs since it permits an application to obtain the expected functionality with fewer  
 92686 statements.
- 92687 The %array and %pointer declarations were added as a compromise between historical systems.  
 92688 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in  
 92689 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements  
 92690 are available for some scanners. Most historical programs should require no change in porting  
 92691 from one system to another because the string being referenced is null-terminated in both cases.  
 92692 (The method used by *flex* in its case is to null-terminate the token in place by remembering the  
 92693 character that used to come right after the token and replacing it before continuing on to the next  
 92694 scan.) Multi-file programs with external references to *yytext* outside the scanner source file  
 92695 should continue to operate on their historical systems, but would require one of the new  
 92696 declarations to be considered strictly portable.
- 92697 The description of EREs avoids unnecessary duplication of ERE details because their meanings  
 92698 within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-2008.
- 92699 The reason for the undefined condition associated with text beginning with a <blank> or within  
 92700 "% { " and "% } " delimiter lines appearing in the *Rules* section is historical practice. Both the BSD  
 92701 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the  
 92702 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*  
 92703 statement). In some cases, the System V *lex* generates an error message or a syntax error,  
 92704 depending on the form of indented input.
- 92705 The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those  
 92706 that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a  
 92707 conforming application.
- 92708 The descriptions of standard output and standard error are somewhat complicated because  
 92709 historical *lex* implementations chose to issue diagnostic messages to standard output (unless *-t*  
 92710 was given). POSIX.1-2008 allows this behavior, but leaves an opening for the more expected  
 92711 behavior of using standard error for diagnostics. Also, the System V behavior of writing the  
 92712 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The

- 92713 programmer can always precisely obtain the desired results by using either the `-t` or `-n` options.
- 92714 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all  
92715 historical implementations support such usage for any of the *file* operands.
- 92716 A description of the *translation table* was deleted from early proposals because of its relatively  
92717 low usage in historical applications.
- 92718 The change to the definition of the *input()* function that allows buffering of input presents the  
92719 opportunity for major performance gains in some applications.
- 92720 The following examples clarify the differences between *lex* regular expressions and regular  
92721 expressions appearing elsewhere in this volume of POSIX.1-2008. For regular expressions of the  
92722 form "*r/x*", the string matching *r* is always returned; confusion may arise when the beginning  
92723 of *x* matches the trailing portion of *r*. For example, given the regular expression "*a\*b/cc*" and  
92724 the input "*aaabcc*", *yytext* would contain the string "*aaab*" on this match. But given the  
92725 regular expression "*x\*/xY*" and the input "*xxxY*", the token *xxx*, not *xx*, is returned by some  
92726 implementations because *xxx* matches "*x\**".
- 92727 In the rule "*ab\*/bc*", the "*b\**" at the end of *r* extends *r*'s match into the beginning of the  
92728 trailing context, so the result is unspecified. If this rule were "*ab/bc*", however, the rule  
92729 matches the text "*ab*" when it is followed by the text "*bc*". In this latter case, the matching of *r*  
92730 cannot extend into the beginning of *x*, so the result is specified.
- 92731 **FUTURE DIRECTIONS**
- 92732 None.
- 92733 **SEE ALSO**
- 92734 *c99*, *ed*, *yacc*
- 92735 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on  
92736 page 215)
- 92737 **CHANGE HISTORY**
- 92738 First released in Issue 2.
- 92739 **Issue 6**
- 92740 This utility is marked as part of the C-Language Development Utilities option.
- 92741 The obsolescent `-c` option is removed.
- 92742 The normative text is reworded to avoid use of the term "must" for application requirements.
- 92743 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing  
92744 behavior on systems with bytes consisting of more than eight bits.
- 92745 **Issue 7**
- 92746 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for  
92747 generated code to conform to the ISO C standard.
- 92748 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language  
92749 trigraphs and curly brace preprocessing tokens.
- 92750 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
92751 apply.
- 92752 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92753 **NAME**92754 link — call *link()* function92755 **SYNOPSIS**92756 XSI `link file1 file2`92757 **DESCRIPTION**92758 The *link* utility shall perform the function call:92759 `link(file1, file2);`92760 A user may need appropriate privileges to invoke the *link* utility.92761 **OPTIONS**

92762 None.

92763 **OPERANDS**

92764 The following operands shall be supported:

92765 *file1* The pathname of an existing file.92766 *file2* The pathname of the new directory entry to be created.92767 **STDIN**

92768 Not used.

92769 **INPUT FILES**

92770 Not used.

92771 **ENVIRONMENT VARIABLES**92772 The following environment variables shall affect the execution of *link*:92773 *LANG* Provide a default value for the internationalization variables that are unset or null.  
92774 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
92775 variables used to determine the values of locale categories.)92776 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
92777 internationalization variables.92778 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
92779 characters (for example, single-byte as opposed to multi-byte characters in  
92780 arguments).92781 *LC\_MESSAGES*  
92782 Determine the locale that should be used to affect the format and contents of  
92783 diagnostic messages written to standard error.92784 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.92785 **ASYNCHRONOUS EVENTS**

92786 Default.

92787 **STDOUT**

92788 None.

92789 **STDERR**

92790 The standard error shall be used only for diagnostic messages.

**link**

Utilities

92791 **OUTPUT FILES**

92792 None.

92793 **EXTENDED DESCRIPTION**

92794 None.

92795 **EXIT STATUS**

92796 The following exit values shall be returned:

92797 0 Successful completion.

92798 &gt;0 An error occurred.

92799 **CONSEQUENCES OF ERRORS**

92800 Default.

92801 **APPLICATION USAGE**

92802 None.

92803 **EXAMPLES**

92804 None.

92805 **RATIONALE**

92806 None.

92807 **FUTURE DIRECTIONS**

92808 None.

92809 **SEE ALSO**92810 *ln*, *unlink*

92811 XBD Chapter 8 (on page 173)

92812 XSH *link()*92813 **CHANGE HISTORY**

92814 First released in Issue 5.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

92815 **NAME**

92816 ln — link files

92817 **SYNOPSIS**92818 ln [-fs] [-L|-P] *source\_file target\_file*92819 ln [-fs] [-L|-P] *source\_file... target\_dir*92820 **DESCRIPTION**

92821 In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination  
 92822 path specified by the *target\_file* operand. If the *-s* option is specified, a symbolic link shall be  
 92823 created for the file specified by the *source\_file* operand. This first synopsis form shall be assumed  
 92824 when the final operand does not name an existing directory; if more than two operands are  
 92825 specified and the final is not an existing directory, an error shall result.

92826 In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the *-s*  
 92827 option is specified a symbolic link, for each file specified by a *source\_file* operand, at a  
 92828 destination path in the existing directory named by *target\_dir*.

92829 If the last operand specifies an existing file of a type not specified by the System Interfaces  
 92830 volume of POSIX.1-2008, the behavior is implementation-defined.

92831 The corresponding destination path for each *source\_file* shall be the concatenation of the target  
 92832 directory pathname, a <slash> character if the target directory pathname did not end in a  
 92833 <slash>, and the last pathname component of the *source\_file*. The second synopsis form shall be  
 92834 assumed when the final operand names an existing directory.

92835 For each *source\_file*:

- 92836 1. If the destination path exists and was created by a previous step, it is unspecified whether  
 92837 *ln* shall write a diagnostic message to standard error, do nothing more with the current  
 92838 *source\_file*, and go on to any remaining *source\_files*; or will continue processing the current  
 92839 *source\_file*. If the destination path exists:
  - 92840 a. If the *-f* option is not specified, *ln* shall write a diagnostic message to standard  
 92841 error, do nothing more with the current *source\_file*, and go on to any remaining  
 92842 *source\_files*.
  - 92843 b. If *destination* names the same directory entry as the current *source\_file* *ln* shall write  
 92844 a diagnostic message to standard error, do nothing more with the current  
 92845 *source\_file*, and go on to any remaining *source\_files*.
  - 92846 c. Actions shall be performed equivalent to the *unlink()* function defined in the  
 92847 System Interfaces volume of POSIX.1-2008, called using *destination* as the *path*  
 92848 argument. If this fails for any reason, *ln* shall write a diagnostic message to  
 92849 standard error, do nothing more with the current *source\_file*, and go on to any  
 92850 remaining *source\_files*.
- 92851 2. If the *-s* option is specified, *ln* shall create a symbolic link named by the destination path  
 92852 and containing as its pathname *source\_file*. The *ln* utility shall do nothing more with  
 92853 *source\_file* and shall go on to any remaining files.
- 92854 3. If *source\_file* is a symbolic link:
  - 92855 a. If the *-P* option is in effect, actions shall be performed equivalent to the *linkat()*  
 92856 function with *source\_file* as the *path1* argument, the destination path as the *path2*  
 92857 argument, *AT\_FDCWD* as the *fd1* and *fd2* arguments, and zero as the *flag*  
 92858 argument.

92859                   b. If the `-L` option is in effect, actions shall be performed equivalent to the `linkat()`  
 92860                   function with `source_file` as the `path1` argument, the destination path as the `path2`  
 92861                   argument, `AT_FDCWD` as the `fd1` and `fd2` arguments, and  
 92862                   `AT_SYMLINK_FOLLOW` as the `flag` argument.

92863                   The `ln` utility shall do nothing more with `source_file` and shall go on to any remaining files.

92864                   4. Actions shall be performed equivalent to the `link()` function defined in the System  
 92865                   Interfaces volume of POSIX.1-2008 using `source_file` as the `path1` argument, and the  
 92866                   destination path as the `path2` argument.

#### 92867 OPTIONS

92868                   The `ln` utility shall conform to XBD Section 12.2 (on page 215).

92869                   The following options shall be supported:

- 92870                   **-f**               Force existing destination pathnames to be removed to allow the link.
- 92871                   **-L**               For each `source_file` operand that names a file of type symbolic link, create a (hard)  
 92872                   link to the file referenced by the symbolic link.
- 92873                   **-P**               For each `source_file` operand that names a file of type symbolic link, create a (hard)  
 92874                   link to the symbolic link itself.
- 92875                   **-s**               Create symbolic links instead of hard links. If the `-s` option is specified, the `-L` and  
 92876                   **-P** options shall be silently ignored.

92877                   Specifying more than one of the mutually-exclusive options `-L` and `-P` shall not be considered  
 92878                   an error. The last option specified shall determine the behavior of the utility (unless the `-s`  
 92879                   option causes it to be ignored).

92880                   If the `-s` option is not specified and neither a `-L` nor a `-P` option is specified, it is  
 92881                   implementation-defined which of the `-L` and `-P` options will be used as the default.

#### 92882 OPERANDS

92883                   The following operands shall be supported:

- 92884                   `source_file`      A pathname of a file to be linked. If the `-s` option is specified, no restrictions on the  
 92885                   type of file or on its existence shall be made. If the `-s` option is not specified,  
 92886                   whether a directory can be linked is implementation-defined.
- 92887                   `target_file`      The pathname of the new directory entry to be created.
- 92888                   `target_dir`      A pathname of an existing directory in which the new directory entries are created.

#### 92889 STDIN

92890                   Not used.

#### 92891 INPUT FILES

92892                   None.

#### 92893 ENVIRONMENT VARIABLES

92894                   The following environment variables shall affect the execution of `ln`:

- 92895                   **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 92896                   (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 92897                   variables used to determine the values of locale categories.)
- 92898                   **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
 92899                   internationalization variables.

|       |                               |                                                                                                                                                                                                                                                                                                                                                   |
|-------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 92900 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                                                                                                                         |
| 92901 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92902 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92903 | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                                                                   |
| 92904 |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                      |
| 92905 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92906 | XSI <i>NLSPATH</i>            | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                             |
| 92907 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                   |
| 92908 |                               | Default.                                                                                                                                                                                                                                                                                                                                          |
| 92909 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                   |
| 92910 |                               | Not used.                                                                                                                                                                                                                                                                                                                                         |
| 92911 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                   |
| 92912 |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                    |
| 92913 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                   |
| 92914 |                               | None.                                                                                                                                                                                                                                                                                                                                             |
| 92915 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                   |
| 92916 |                               | None.                                                                                                                                                                                                                                                                                                                                             |
| 92917 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                   |
| 92918 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                      |
| 92919 | 0                             | All the specified files were linked successfully.                                                                                                                                                                                                                                                                                                 |
| 92920 | >0                            | An error occurred.                                                                                                                                                                                                                                                                                                                                |
| 92921 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                   |
| 92922 |                               | Default.                                                                                                                                                                                                                                                                                                                                          |
| 92923 | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                                                                                                   |
| 92924 |                               | None.                                                                                                                                                                                                                                                                                                                                             |
| 92925 | <b>EXAMPLES</b>               |                                                                                                                                                                                                                                                                                                                                                   |
| 92926 |                               | None.                                                                                                                                                                                                                                                                                                                                             |
| 92927 | <b>RATIONALE</b>              |                                                                                                                                                                                                                                                                                                                                                   |
| 92928 |                               | The CONSEQUENCES OF ERRORS section does not require <i>ln -f a b</i> to remove <i>b</i> if a subsequent link operation would fail.                                                                                                                                                                                                                |
| 92929 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92930 |                               | Some historic versions of <i>ln</i> (including the one specified by the SVID) unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions prompt for confirmation before attempting the unlink. In these versions the <i>-f</i> option causes <i>ln</i> not to attempt to prompt for confirmation. |
| 92931 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92932 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92933 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92934 |                               | This allows <i>ln</i> to succeed in creating links when the target file already exists, even if the file itself is not writable (although the directory must be). Early proposals specified this functionality.                                                                                                                                   |
| 92935 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92936 |                               | This volume of POSIX.1-2008 does not allow the <i>ln</i> utility to unlink existing destination paths by default for the following reasons:                                                                                                                                                                                                       |
| 92937 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92938 |                               | • The <i>ln</i> utility has historically been used to provide locking for shell applications, a usage that is incompatible with <i>ln</i> unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.                                                                                 |
| 92939 |                               |                                                                                                                                                                                                                                                                                                                                                   |
| 92940 |                               |                                                                                                                                                                                                                                                                                                                                                   |

- 92941 • This functionality gave *ln* the ability to destroy the link structure of files, which changes
- 92942 the historical behavior of *ln*.
- 92943 • This functionality is easily replicated with a combination of *rm* and *ln*.
- 92944 • It is not historical practice in many systems; BSD and BSD-derived systems do not support
- 92945 this behavior. Unfortunately, whichever behavior is selected can cause scripts written
- 92946 expecting the other behavior to fail.
- 92947 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not
- 92948 permit the target to exist already.

92949 This volume of POSIX.1-2008 retains the `-f` option to provide support for shell scripts depending  
 92950 on the SVID semantics. It seems likely that shell scripts would not be written to handle  
 92951 prompting by *ln* and would therefore have specified the `-f` option.

92952 The `-f` option is an undocumented feature of many historical versions of the *ln* utility, allowing  
 92953 linking to directories. These versions require modification.

92954 Early proposals of this volume of POSIX.1-2008 also required a `-i` option, which behaved like the  
 92955 `-i` options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not  
 92956 historical practice for the *ln* utility and has been omitted.

92957 The `-L` and `-P` options allow for implementing both common behaviors of the *ln* utility. Earlier  
 92958 versions of this standard did not specify these options and required the behavior now described  
 92959 for the `-L` option. Many systems by default or as an alternative provided a non-conforming *ln*  
 92960 utility with the behavior now described for the `-P` option. Since applications could not rely on *ln*  
 92961 following links in practice, the `-L` and `-P` options were added to specify the desired behavior for  
 92962 the application.

92963 The `-L` and `-P` options are ignored when `-s` is specified in order to allow an alias to be created to  
 92964 alter the default behavior when creating hard links (for example, *alias ln='ln -L'*). They serve no  
 92965 purpose when `-s` is specified, since *source\_file* is then just a string to be used as the contents of  
 92966 the created symbolic link and need not exist as a file.

92967 The specification ensures that *ln a a* with or without the `-f` option will not unlink the file *a*.  
 92968 Earlier versions of this standard were unclear in this case.

#### 92969 FUTURE DIRECTIONS

92970 None.

#### 92971 SEE ALSO

92972 *chmod*, *find*, *pax*, *rm*

92973 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

92974 XSH *lnk()*, *unlink()*

#### 92975 CHANGE HISTORY

92976 First released in Issue 2.

#### 92977 Issue 6

92978 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b  
 92979 draft standard.

#### 92980 Issue 7

92981 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

92982 SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

92983 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92984  
92985

The **-L** and **-P** options are added to make it implementation-defined whether the *ln* utility follows symbolic links.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**locale**

Utilities

92986 **NAME**

92987 locale — get locale-specific information

92988 **SYNOPSIS**

92989 locale [-a|-m]

92990 locale [-ck] name...

92991 **DESCRIPTION**

92992 The *locale* utility shall write information about the current locale environment, or all public  
 92993 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by  
 92994 the implementation that is accessible to the application.

92995 When *locale* is invoked without any arguments, it shall summarize the current locale  
 92996 environment for each locale category as determined by the settings of the environment variables  
 92997 defined in XBD Chapter 7 (on page 135).

92998 When invoked with operands, it shall write values that have been assigned to the keywords in  
 92999 the locale categories, as follows:

- 93000 • Specifying a keyword name shall select the named keyword and the category containing  
 93001 that keyword.
- 93002 • Specifying a category name shall select the named category and all keywords in that  
 93003 category.

93004 **OPTIONS**93005 The *locale* utility shall conform to XBD Section 12.2 (on page 215).

93006 The following options shall be supported:

- 93007 **-a** Write information about all available public locales. The available locales shall  
 93008 include **POSIX**, representing the POSIX locale. The manner in which the  
 93009 implementation determines what other locales are available is implementation-  
 93010 defined.
- 93011 **-c** Write the names of selected locale categories; see the STDOUT section. The **-c**  
 93012 option increases readability when more than one category is selected (for example,  
 93013 via more than one keyword name or via a category name). It is valid both with  
 93014 and without the **-k** option.
- 93015 **-k** Write the names and values of selected keywords. The implementation may omit  
 93016 values for some keywords; see the OPERANDS section.
- 93017 **-m** Write names of available charmaps; see XBD Section 6.1 (on page 125).

93018 **OPERANDS**

93019 The following operand shall be supported:

- 93020 *name* The name of a locale category as defined in XBD Chapter 7 (on page 135), the name  
 93021 of a keyword in a locale category, or the reserved name **charmap**. The named  
 93022 category or keyword shall be selected for output. If a single *name* represents both a  
 93023 locale category name and a keyword name in the current locale, the results are  
 93024 unspecified. Otherwise, both category and keyword names can be specified as  
 93025 *name* operands, in any sequence. It is implementation-defined whether any  
 93026 keyword values are written for the categories *LC\_CTYPE* and *LC\_COLLATE*.

93027 **STDIN**

93028 Not used.

93029 **INPUT FILES**

93030 None.

93031 **ENVIRONMENT VARIABLES**93032 The following environment variables shall affect the execution of *locale*:

93033 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 93034 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 93035 variables used to determine the values of locale categories.)

93036 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 93037 internationalization variables.

93038 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 93039 characters (for example, single-byte as opposed to multi-byte characters in  
 93040 arguments and input files).

93041 *LC\_MESSAGES*

93042 Determine the locale that should be used to affect the format and contents of  
 93043 diagnostic messages written to standard error.

93044 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

93045 XSI The application shall ensure that the *LANG*, *LC\_\**, and *NLSPATH* environment variables specify  
 93046 the current locale environment to be written out; they shall be used if the *-a* option is not  
 93047 specified.

93048 **ASYNCHRONOUS EVENTS**

93049 Default.

93050 **STDOUT**93051 The *LANG* variable shall be written first using the format:

93052 "LANG=%s\n", &lt;value&gt;

93053 If *LANG* is not set or is an empty string, the value is the empty string.

93054 If *locale* is invoked without any options or operands, the names and values of the *LC\_\**  
 93055 environment variables described in this volume of POSIX.1-2008 shall be written to the standard  
 93056 output, one variable per line, and each line using the following format. Only those variables set  
 93057 in the environment and not overridden by *LC\_ALL* shall be written using this format:

93058 "%s=%s\n", &lt;variable\_name&gt;, &lt;value&gt;

93059 The names of those *LC\_\** variables associated with locale categories defined in this volume of  
 93060 POSIX.1-2008 that are not set in the environment or are overridden by *LC\_ALL* shall be written  
 93061 in the following format:

93062 "%s=\"%s\"\n", &lt;variable\_name&gt;, &lt;implied value&gt;

93063 The <implied value> shall be the name of the locale that has been selected for that category by the  
 93064 implementation, based on the values in *LANG* and *LC\_ALL*, as described in XBD Chapter 8 (on  
 93065 page 173).

93066 The <value> and <implied value> shown above shall be properly quoted for possible later reentry  
 93067 to the shell. The <value> shall not be quoted using double-quotes (so that it can be distinguished  
 93068 by the user from the <implied value> case, which always requires double-quotes).

93069 The *LC\_ALL* variable shall be written last, using the first format shown above. If it is not set, it  
93070 shall be written as:

93071 "LC\_ALL=\n"

93072 If any arguments are specified:

93073 1. If the *-a* option is specified, the names of all the public locales shall be written, each in the  
93074 following format:

93075 "%s\n", <locale name>

93076 2. If the *-c* option is specified, the names of all selected categories shall be written, each in  
93077 the following format:

93078 "%s\n", <category name>

93079 If keywords are also selected for writing (see following items), the category name output  
93080 shall precede the keyword output for that category.

93081 If the *-c* option is not specified, the names of the categories shall not be written; only the  
93082 keywords, as selected by the <name> operand, shall be written.

93083 3. If the *-k* option is specified, the names and values of selected keywords shall be written.  
93084 If a value is non-numeric and is not a compound keyword value, it shall be written in the  
93085 following format:

93086 "%s=\"%s\"\n", <keyword name>, <keyword value>

93087 If a value is a non-numeric compound keyword value, it shall either be written in the  
93088 format:

93089 "%s=\"%s\"\n", <keyword name>, <keyword value>

93090 where the <keyword value> is a single string of values separated by <semicolon>  
93091 characters, or it shall be written in the format:

93092 "%s=%s\n", <keyword name>, <keyword value>

93093 where the <keyword value> is encoded as a set of strings, each enclosed in double-  
93094 quotation-marks, separated by <semicolon> characters.

93095 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
93096 *localedef -f* option when the locale was created shall be written, with the word **charmap** as  
93097 <keyword name>.

93098 If a value is numeric, it shall be written in one of the following formats:

93099 "%s=%d\n", <keyword name>, <keyword value>

93100 "%s=%c%o\n", <keyword name>, <escape character>, <keyword value>

93101 "%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>

93102 where the <escape character> is that identified by the **escape\_char** keyword in the current  
93103 locale; see XBD Section 7.3 (on page 136).

93104 Compound keyword values (list entries) shall be separated in the output by <semicolon>  
93105 characters. When included in keyword values, the <semicolon>, <backslash>, double-  
93106 quote, and any control character shall be preceded (escaped) with the escape character.

93107 4. If the **-k** option is not specified, selected keyword values shall be written, each in the  
93108 following format:

93109 "%s\n", <keyword value>

93110 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
93111 *localedef* **-f** option when the locale was created shall be written.

93112 5. If the **-m** option is specified, then a list of all available charmaps shall be written, each in  
93113 the format:

93114 "%s\n", <charmap>

93115 where <charmap> is in a format suitable for use as the option-argument to the *localedef* **-f**  
93116 option.

#### 93117 **STDERR**

93118 The standard error shall be used only for diagnostic messages.

#### 93119 **OUTPUT FILES**

93120 None.

#### 93121 **EXTENDED DESCRIPTION**

93122 None.

#### 93123 **EXIT STATUS**

93124 The following exit values shall be returned:

93125 0 All the requested information was found and output successfully.

93126 >0 An error occurred.

#### 93127 **CONSEQUENCES OF ERRORS**

93128 Default.

#### 93129 **APPLICATION USAGE**

93130 If the *LANG* environment variable is not set or set to an empty value, or one of the *LC\_\**  
93131 environment variables is set to an unrecognized value, the actual locales assumed (if any) are  
93132 implementation-defined as described in XBD Chapter 8 (on page 173).

93133 Implementations are not required to write out the actual values for keywords in the categories  
93134 *LC\_CTYPE* and *LC\_COLLATE*; however, they must write out the categories (allowing an  
93135 application to determine, for example, which character classes are available).

#### 93136 **EXAMPLES**

93137 In the following examples, the assumption is that locale environment variables are set as  
93138 follows:

93139 *LANG*=locale\_x

93140 *LC\_COLLATE*=locale\_y

93141 The command *locale* would result in the following output:

93142 *LANG*=locale\_x

93143 *LC\_CTYPE*="locale\_x"

93144 *LC\_COLLATE*=locale\_y

93145 *LC\_TIME*="locale\_x"

93146 *LC\_NUMERIC*="locale\_x"

93147 *LC\_MONETARY*="locale\_x"

93148 *LC\_MESSAGES*="locale\_x"

93149 *LC\_ALL*=

93150 The order of presentation of the categories is not specified by this volume of POSIX.1-2008.

93151 The command:

```
93152 LC_ALL=POSIX locale -ck decimal_point
```

93153 would produce:

```
93154 LC_NUMERIC
93155 decimal_point="."
```

93156 The following command shows an application of *locale* to determine whether a user-supplied  
93157 response is affirmative:

```
93158 if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
93159 then
93160     affirmative processing goes here
93161 else
93162     non-affirmative processing goes here
93163 fi
```

#### 93164 RATIONALE

93165 The output for categories *LC\_CTYPE* and *LC\_COLLATE* has been made implementation-defined  
93166 because there is a questionable value in having a shell script receive an entire array of characters.  
93167 It is also difficult to return a logical collation description, short of returning a complete *localedef*  
93168 source.

93169 The **-m** option was included to allow applications to query for the existence of charmaps. The  
93170 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the  
93171 system.

93172 The **-c** option was included for readability when more than one category is selected (for  
93173 example, via more than one keyword name or via a category name). It is valid both with and  
93174 without the **-k** option.

93175 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the  
93176 current locale was created, was included to allow applications needing the information to  
93177 retrieve it.

#### 93178 FUTURE DIRECTIONS

93179 None.

#### 93180 SEE ALSO

93181 *localedef*

93182 XBD Section 6.1 (on page 125), Chapter 7 (on page 135), Chapter 8 (on page 173), Section 12.2 (on  
93183 page 215)

#### 93184 CHANGE HISTORY

93185 First released in Issue 4.

#### 93186 Issue 5

93187 The FUTURE DIRECTIONS section is added.

#### 93188 Issue 6

93189 The normative text is reworded to avoid use of the term “must” for application requirements.

93190 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error  
93191 in the STDOUT section.

93192 **Issue 7**

93193 Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the  
93194 standard output for the `-k` option when `LANG` is not set or is an empty string.

93195 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## localedef

Utilities

## 93196 NAME

93197 localedef — define locale environment

## 93198 SYNOPSIS

93199 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code\_set\_name*] *name*

## 93200 DESCRIPTION

93201 The *localedef* utility shall convert source definitions for locale categories into a format usable by  
 93202 the functions and utilities whose operational behavior is determined by the setting of the locale  
 93203 environment variables defined in XBD Chapter 7 (on page 135). It is implementation-defined  
 93204 whether users have the capability to create new locales, in addition to those supplied by the  
 93205 implementation. If the symbolic constant POSIX2\_LOCALEDEF is defined, the system supports  
 93206 XSI the creation of new locales. On XSI-conformant systems, the symbolic constant  
 93207 POSIX2\_LOCALEDEF shall be defined.

93208 The utility shall read source definitions for one or more locale categories belonging to the same  
 93209 locale from the file named in the *-i* option (if specified) or from standard input.

93210 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or  
 93211 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may  
 93212 restrict the capability to create or modify public locales to users with appropriate privileges.

93213 Each category source definition shall be identified by the corresponding environment variable  
 93214 name and terminated by an **END** *category-name* statement. The following categories shall be  
 93215 supported. In addition, the input may contain source for implementation-defined categories.

93216 *LC\_CTYPE* Defines character classification and case conversion.

93217 *LC\_COLLATE*

93218 Defines collation rules.

93219 *LC\_MONETARY*

93220 Defines the format and symbols used in formatting of monetary information.

93221 *LC\_NUMERIC*

93222 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary  
 93223 numeric editing.

93224 *LC\_TIME* Defines the format and content of date and time information.

93225 *LC\_MESSAGES*

93226 Defines the format and values of affirmative and negative responses.

## 93227 OPTIONS

93228 The *localedef* utility shall conform to XBD Section 12.2 (on page 215).

93229 The following options shall be supported:

93230 *-c* Create permanent output even if warning messages have been issued.

93231 *-f charmap* Specify the pathname of a file containing a mapping of character symbols and  
 93232 collating element symbols to actual character encodings. The format of the  
 93233 *charmap* is described in XBD Section 6.4 (on page 129). The application shall ensure  
 93234 that this option is specified if symbolic names (other than collating symbols  
 93235 defined in a **collating-symbol** keyword) are used. If the *-f* option is not present, an  
 93236 implementation-defined character mapping shall be used.

93237 *-i inputfile* The pathname of a file containing the source definitions. If this option is not  
 93238 present, source definitions shall be read from standard input. The format of the  
 93239 *inputfile* is described in XBD Section 7.3 (on page 136).

- 93240            **-u** *code\_set\_name*
- 93241                   Specify the name of a codeset used as the target mapping of character symbols and
- 93242                   collating element symbols whose encoding values are defined in terms of the
- 93243                   ISO/IEC 10646-1: 2000 standard position constant values.
- 93244   **OPERANDS**
- 93245            The following operand shall be supported:
- 93246            *name*           Identifies the locale; see XBD Chapter 7 (on page 135) for a description of the use of
- 93247                   this name. If the name contains one or more <slash> characters, *name* shall be
- 93248                   interpreted as a pathname where the created locale definitions shall be stored. If
- 93249                   *name* does not contain any <slash> characters, the interpretation of the name is
- 93250                   implementation-defined and the locale shall be public. The ability to create public
- 93251                   locales in this way may be restricted to users with appropriate privileges. (As a
- 93252                   consequence of specifying one *name*, although several categories can be processed
- 93253                   in one execution, only categories belonging to the same locale can be processed.)
- 93254   **STDIN**
- 93255            Unless the **-i** option is specified, the standard input shall be a text file containing one or more
- 93256            locale category source definitions, as described in XBD Section 7.3 (on page 136). When lines are
- 93257            continued using the escape character mechanism, there is no limit to the length of the
- 93258            accumulated continued line.
- 93259   **INPUT FILES**
- 93260            The character set mapping file specified as the *charmap* option-argument is described in XBD
- 93261            Section 6.4 (on page 129). If a locale category source definition contains a **copy** statement, as
- 93262            defined in XBD Chapter 7 (on page 135), and the **copy** statement names a valid, existing locale,
- 93263            then *localedef* shall behave as if the source definition had contained a valid category source
- 93264            definition for the named locale.
- 93265   **ENVIRONMENT VARIABLES**
- 93266            The following environment variables shall affect the execution of *localedef*.
- 93267            **LANG**            Provide a default value for the internationalization variables that are unset or null.
- 93268                   (See XBD Section 8.2 (on page 174) for the precedence of internationalization
- 93269                   variables used to determine the values of locale categories.)
- 93270            **LC\_ALL**        If set to a non-empty string value, override the values of all the other
- 93271                   internationalization variables.
- 93272            **LC\_COLLATE**
- 93273                   (This variable has no affect on *localedef*; the POSIX locale is used for this category.)
- 93274            **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as
- 93275                   characters (for example, single-byte as opposed to multi-byte characters in
- 93276                   arguments and input files). This variable has no affect on the processing of *localedef*
- 93277                   input data; the POSIX locale is used for this purpose, regardless of the value of this
- 93278                   variable.
- 93279            **LC\_MESSAGES**
- 93280                   Determine the locale that should be used to affect the format and contents of
- 93281                   diagnostic messages written to standard error.
- 93282    XSI        **NLSPATH**    Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

**localedef**

Utilities

93283 **ASYNCHRONOUS EVENTS**

93284 Default.

93285 **STDOUT**

93286 The utility shall report all categories successfully processed, in an unspecified format.

93287 **STDERR**

93288 The standard error shall be used only for diagnostic messages.

93289 **OUTPUT FILES**93290 The format of the created output is unspecified. If the *name* operand does not contain a `/`,  
93291 the existence of an output file for the locale is unspecified.93292 **EXTENDED DESCRIPTION**93293 When the `-u` option is used, the *code\_set\_name* option-argument shall be interpreted as an  
93294 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard  
93295 position constant values shall be converted via an implementation-defined method. Both the  
93296 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal,  
93297 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset  
93298 represented by the implementation-defined name can be any codeset that is supported by the  
93299 implementation.93300 When conflicts occur between the *charmap* specification of `<code_set_name>`, `<mb_cur_max>`, or  
93301 `<mb_cur_min>` and the implementation-defined interpretation of these respective items for the  
93302 codeset represented by the `-u` option-argument *code\_set\_name*, the result is unspecified.93303 When conflicts occur between the *charmap* encoding values specified for symbolic names of  
93304 characters of the portable character set and the implementation-defined assignment of character  
93305 encoding values, the result is unspecified.93306 If a non-printable character in the *charmap* has a width specified that is not `-1`, the result will be  
93307 undefined.93308 **EXIT STATUS**

93309 The following exit values shall be returned:

93310 0 No errors occurred and the locales were successfully created.

93311 1 Warnings occurred and the locales were successfully created.

93312 2 The locale specification exceeded implementation limits or the coded character set or sets  
93313 used were not supported by the implementation, and no locale was created.

93314 3 The capability to create new locales is not supported by the implementation.

93315 &gt;3 Warnings or errors occurred and no output was created.

93316 **CONSEQUENCES OF ERRORS**

93317 If an error is detected, no permanent output shall be created.

93318 If warnings occur, permanent output shall be created if the `-c` option was specified. The  
93319 following conditions shall cause warning messages to be issued:93320 • If a symbolic name not found in the *charmap* file is used for the descriptions of the  
93321 *LC\_CTYPE* or *LC\_COLLATE* categories (for other categories, this shall be an error  
93322 condition).93323 • If the number of operands to the **order** keyword exceeds the `{COLL_WEIGHTS_MAX}`  
93324 limit.

- 93325 • If optional keywords not supported by the implementation are present in the source.

93326 Other implementation-defined conditions may also cause warnings.

#### 93327 APPLICATION USAGE

93328 The *charmap* definition is optional, and is contained outside the locale definition. This allows  
 93329 both completely self-defined source files, and generic sources (applicable to more than one  
 93330 codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the  
 93331 portable character set. As explained in XBD Section 6.4 (on page 129), it is implementation-  
 93332 defined whether or not users or applications can provide additional character set description  
 93333 files. Therefore, the `-f` option might be operable only when an implementation-defined *charmap*  
 93334 is named.

#### 93335 EXAMPLES

93336 None.

#### 93337 RATIONALE

93338 The output produced by the *localedef* utility is implementation-defined. The *name* operand is  
 93339 used to identify the specific locale. (As a consequence, although several categories can be  
 93340 processed in one execution, only categories belonging to the same locale can be processed.)

#### 93341 FUTURE DIRECTIONS

93342 None.

#### 93343 SEE ALSO

93344 *locale*

93345 XBD Section 6.4 (on page 129), Chapter 7 (on page 135), Chapter 8 (on page 173), Section 12.2 (on  
 93346 page 215)

#### 93347 CHANGE HISTORY

93348 First released in Issue 4.

#### 93349 Issue 6

93350 The `-u` option is added, as specified in the IEEE P1003.2b draft standard.

93351 The normative text is reworded to avoid use of the term “must” for application requirements.

93352 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the  
 93353 OPERANDS section describing the ability to create public locales.

93354 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent  
 93355 with the descriptions of **WIDTH** and **WIDTH\_DEFAULT** in the Base Definitions volume of  
 93356 POSIX.1-2008.

#### 93357 Issue 7

93358 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**logger**

Utilities

93359 **NAME**93360 `logger` — log messages93361 **SYNOPSIS**93362 `logger string...`93363 **DESCRIPTION**

93364 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*  
 93365 operands provided by the user. The messages are expected to be evaluated later by personnel  
 93366 performing system administration tasks.

93367 It is implementation-defined whether messages written in locales other than the POSIX locale  
 93368 are effective.

93369 **OPTIONS**

93370 None.

93371 **OPERANDS**

93372 The following operand shall be supported:

93373 *string* One of the string arguments whose contents are concatenated together, in the order  
 93374 specified, separated by single <space> characters.

93375 **STDIN**

93376 Not used.

93377 **INPUT FILES**

93378 None.

93379 **ENVIRONMENT VARIABLES**93380 The following environment variables shall affect the execution of *logger*:

93381 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 93382 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 93383 variables used to determine the values of locale categories.)

93384 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 93385 internationalization variables.

93386 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 93387 characters (for example, single-byte as opposed to multi-byte characters in  
 93388 arguments).

93389 *LC\_MESSAGES*

93390 Determine the locale that should be used to affect the format and contents of  
 93391 diagnostic messages written to standard error. (This means diagnostics from *logger*  
 93392 to the user or application, not diagnostic messages that the user is sending to the  
 93393 system administrator.)

93394 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

93395 **ASYNCHRONOUS EVENTS**

93396 Default.

93397 **STDOUT**

93398 Not used.

93399 **STDERR**

93400 The standard error shall be used only for diagnostic messages.

93401 **OUTPUT FILES**

93402 Unspecified.

93403 **EXTENDED DESCRIPTION**

93404 None.

93405 **EXIT STATUS**

93406 The following exit values shall be returned:

93407 0 Successful completion.

93408 >0 An error occurred.

93409 **CONSEQUENCES OF ERRORS**

93410 Default.

93411 **APPLICATION USAGE**

93412 This utility allows logging of information for later use by a system administrator or programmer  
93413 in determining why non-interactive utilities have failed. The locations of the saved messages,  
93414 their format, and retention period are all unspecified. There is no method for a conforming  
93415 application to read messages, once written.

93416 **EXAMPLES**

93417 A batch application, running non-interactively, tries to read a configuration file and fails; it may  
93418 attempt to notify the system administrator with:

93419 `logger myname: unable to read file foo. [timestamp]`

93420 **RATIONALE**

93421 The standard developers believed strongly that some method of alerting administrators to errors  
93422 was necessary. The obvious example is a batch utility, running non-interactively, that is unable to  
93423 read its configuration files or that is unable to create or write its results file. However, the  
93424 standard developers did not wish to define the format or delivery mechanisms as they have  
93425 historically been (and will probably continue to be) very system-specific, as well as involving  
93426 functionality clearly outside the scope of this volume of POSIX.1-2008.

93427 The text with *LC\_MESSAGES* about diagnostic messages means diagnostics from *logger* to the  
93428 user or application, not diagnostic messages that the user is sending to the system administrator.

93429 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

93430 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient  
93431 justification to exclude these utilities from this volume of POSIX.1-2008. It is also arguable that  
93432 they are, in fact, testable, but that the tests themselves are not portable.

93433 **FUTURE DIRECTIONS**

93434 None.

93435 **SEE ALSO**

93436 *lp*, *mailx*, *write*

93437 XBD Chapter 8 (on page 173)

93438 **CHANGE HISTORY**

93439 First released in Issue 4.

## logger

*Utilities*

93440 **Issue 7**  
93441

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

93442 **NAME**

93443 logname — return the user's login name

93444 **SYNOPSIS**

93445 logname

93446 **DESCRIPTION**

93447 The *logname* utility shall write the user's login name to standard output. The login name shall be  
 93448 the string that would be returned by the *getlogin()* function defined in the System Interfaces  
 93449 volume of POSIX.1-2008. Under the conditions where the *getlogin()* function would fail, the  
 93450 *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit  
 93451 status.

93452 **OPTIONS**

93453 None.

93454 **OPERANDS**

93455 None.

93456 **STDIN**

93457 Not used.

93458 **INPUT FILES**

93459 None.

93460 **ENVIRONMENT VARIABLES**93461 The following environment variables shall affect the execution of *logname*:

93462 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 93463 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 93464 variables used to determine the values of locale categories.)

93465 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 93466 internationalization variables.

93467 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 93468 characters (for example, single-byte as opposed to multi-byte characters in  
 93469 arguments).

93470 *LC\_MESSAGES*

93471 Determine the locale that should be used to affect the format and contents of  
 93472 diagnostic messages written to standard error.

93473 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

93474 **ASYNCHRONOUS EVENTS**

93475 Default.

93476 **STDOUT**93477 The *logname* utility output shall be a single line consisting of the user's login name:

93478 "%s\n", &lt;login name&gt;

93479 **STDERR**

93480 The standard error shall be used only for diagnostic messages.

93481 **OUTPUT FILES**

93482 None.

**logname***Utilities*93483 **EXTENDED DESCRIPTION**

93484 None.

93485 **EXIT STATUS**

93486 The following exit values shall be returned:

93487 0 Successful completion.

93488 &gt;0 An error occurred.

93489 **CONSEQUENCES OF ERRORS**

93490 Default.

93491 **APPLICATION USAGE**93492 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment  
93493 changes could produce erroneous results.93494 **EXAMPLES**

93495 None.

93496 **RATIONALE**93497 The **passwd** file is not listed as required because the implementation may have other means of  
93498 mapping login names.93499 **FUTURE DIRECTIONS**

93500 None.

93501 **SEE ALSO**93502 *id*, *who*

93503 XBD Chapter 8 (on page 173)

93504 XSH *getlogin()*93505 **CHANGE HISTORY**

93506 First released in Issue 2.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

93507 **NAME**

93508 lp — send files to a printer

93509 **SYNOPSIS**93510 lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]93511 **DESCRIPTION**

93512 The *lp* utility shall copy the input files to an output destination in an unspecified manner. The  
 93513 default output destination should be to a hardcopy device, such as a printer or microfilm  
 93514 recorder, that produces non-volatile, human-readable documents. If such a device is not  
 93515 available to the application, or if the system provides no such device, the *lp* utility shall exit with  
 93516 a non-zero exit status.

93517 The actual writing to the output device may occur some time after the *lp* utility successfully  
 93518 exits. During the portion of the writing that corresponds to each input file, the implementation  
 93519 shall guarantee exclusive access to the device.

93520 The *lp* utility shall associate a unique *request ID* with each request.

93521 Normally, a banner page is produced to separate and identify each print job. This page may be  
 93522 suppressed by implementation-defined conditions, such as an operator command or one of the  
 93523 **-o option** values.

93524 **OPTIONS**93525 The *lp* utility shall conform to XBD Section 12.2 (on page 215).

93526 The following options shall be supported:

93527 **-c** Exit only after further access to any of the input files is no longer required. The  
 93528 application can then safely delete or modify the files without affecting the output  
 93529 operation. Normally, files are not copied, but are linked whenever possible. If the  
 93530 **-c** option is not given, then the user should be careful not to remove any of the  
 93531 files before the request has been printed in its entirety. It should also be noted that  
 93532 in the absence of the **-c** option, any changes made to the named files after the  
 93533 request is made but before it is printed may be reflected in the printed output. On  
 93534 some implementations, **-c** may be on by default.

93535 **-d *dest*** Specify a string that names the destination (*dest*). If *dest* is a printer, the request  
 93536 shall be printed only on that specific printer. If *dest* is a class of printers, the request  
 93537 shall be printed on the first available printer that is a member of the class. Under  
 93538 certain conditions (printer unavailability, file space limitation, and so on), requests  
 93539 for specific destinations need not be accepted. Destination names vary between  
 93540 systems.

93541 If **-d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable  
 93542 is set, an unspecified destination is used. The **-d *dest*** option shall take precedence  
 93543 over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are  
 93544 undefined when *dest* contains a value that is not a valid destination name.

93545 **-m** Send mail (see *mailx*) after the files have been printed. By default, no mail is sent  
 93546 upon normal completion of the print request.

93547 **-n *copies*** Write *copies* number of copies of the files, where *copies* is a positive decimal integer.  
 93548 The methods for producing multiple copies and for arranging the multiple copies  
 93549 when multiple *file* operands are used are unspecified, except that each file shall be  
 93550 output as an integral whole, not interleaved with portions of other files.

|       |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 93551 | <b>-o option</b>             | Specify printer-dependent or class-dependent <i>options</i> . Several such <i>options</i> may be collected by specifying the <b>-o</b> option more than once.                                                                                                                                                                                                                                                                                    |
| 93552 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93553 | <b>-s</b>                    | Suppress messages from <i>lp</i> .                                                                                                                                                                                                                                                                                                                                                                                                               |
| 93554 | <b>-t title</b>              | Write <i>title</i> on the banner page of the output.                                                                                                                                                                                                                                                                                                                                                                                             |
| 93555 | <b>-w</b>                    | Write a message on the user's terminal after the files have been printed. If the user is not logged in, then mail shall be sent instead.                                                                                                                                                                                                                                                                                                         |
| 93556 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93557 | <b>OPERANDS</b>              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93558 |                              | The following operand shall be supported:                                                                                                                                                                                                                                                                                                                                                                                                        |
| 93559 | <i>file</i>                  | A pathname of a file to be output. If no <i>file</i> operands are specified, or if a <i>file</i> operand is ' - ', the standard input shall be used. If a <i>file</i> operand is used, but the <b>-c</b> option is not specified, the process performing the writing to the output device may have user and group permissions that differ from that of the process invoking <i>lp</i> .                                                          |
| 93560 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93561 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93562 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93563 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93564 | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93565 |                              | The standard input shall be used only if no <i>file</i> operands are specified, or if a <i>file</i> operand is ' - '.                                                                                                                                                                                                                                                                                                                            |
| 93566 |                              | See the INPUT FILES section.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 93567 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93568 |                              | The input files shall be text files.                                                                                                                                                                                                                                                                                                                                                                                                             |
| 93569 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93570 |                              | The following environment variables shall affect the execution of <i>lp</i> :                                                                                                                                                                                                                                                                                                                                                                    |
| 93571 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                                                                               |
| 93572 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93573 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93574 | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                         |
| 93575 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93576 | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                                                        |
| 93577 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93578 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93579 | <i>LC_MESSAGES</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93580 |                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.                                                                                                                                                                                                                                                                 |
| 93581 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93582 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93583 | <i>LC_TIME</i>               | Determine the format and contents of date and time strings displayed in the <i>lp</i> banner page, if any.                                                                                                                                                                                                                                                                                                                                       |
| 93584 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93585 | <i>LPDEST</i>                | Determine the destination. If the <i>LPDEST</i> environment variable is not set, the <i>PRINTER</i> environment variable shall be used. The <b>-d dest</b> option takes precedence over <i>LPDEST</i> . Results are undefined when <b>-d</b> is not specified and <i>LPDEST</i> contains a value that is not a valid destination name.                                                                                                           |
| 93586 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93587 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93588 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93589 | XSI <i>NLSPATH</i>           | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                            |
| 93590 | <i>PRINTER</i>               | Determine the output device or destination. If the <i>LPDEST</i> and <i>PRINTER</i> environment variables are not set, an unspecified output device is used. The <b>-d dest</b> option and the <i>LPDEST</i> environment variable shall take precedence over <i>PRINTER</i> . Results are undefined when <b>-d</b> is not specified, <i>LPDEST</i> is unset, and <i>PRINTER</i> contains a value that is not a valid device or destination name. |
| 93591 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93592 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93593 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93594 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

- 93595            *TZ*            Determine the timezone used to calculate date and time strings displayed in the *lp*  
 93596            banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be  
 93597            used.
- 93598    **ASYNCHRONOUS EVENTS**
- 93599            Default.
- 93600    **STDOUT**
- 93601            The *lp* utility shall write a *request ID* to the standard output, unless *-s* is specified. The format of  
 93602            the message is unspecified. The request ID can be used on systems supporting the historical  
 93603            *cancel* and *lpstat* utilities.
- 93604    **STDERR**
- 93605            The standard error shall be used only for diagnostic messages.
- 93606    **OUTPUT FILES**
- 93607            None.
- 93608    **EXTENDED DESCRIPTION**
- 93609            None.
- 93610    **EXIT STATUS**
- 93611            The following exit values shall be returned:
- 93612            0    All input files were processed successfully.
- 93613            >0   No output device was available, or an error occurred.
- 93614    **CONSEQUENCES OF ERRORS**
- 93615            Default.
- 93616    **APPLICATION USAGE**
- 93617            The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's  
 93618            default page size.
- 93619            A conforming application can use one of the *file* operands only with the *-c* option or if the file is  
 93620            publicly readable and guaranteed to be available at the time of printing. This is because  
 93621            POSIX.1-2008 gives the implementation the freedom to queue up the request for printing at  
 93622            some later time by a different process that might not be able to access the file.
- 93623    **EXAMPLES**
- 93624            1. To print file *file*:
- 93625            `lp -c file`
- 93626            2. To print multiple files with headers:
- 93627            `pr file1 file2 | lp`
- 93628    **RATIONALE**
- 93629            The *lp* utility was designed to be a basic version of a utility that is already available in many  
 93630            historical implementations. The standard developers considered that it should be implementable  
 93631            simply as:
- 93632            `cat "$@" > /dev/lp`
- 93633            after appropriate processing of options, if that is how the implementation chose to do it and if  
 93634            exclusive access could be granted (so that two users did not write to the device simultaneously).  
 93635            Although in the future the standard developers may add other options to this utility, it should  
 93636            always be able to execute with no options or operands and send the standard input to an

- 93637 unspecified output device.
- 93638 This volume of POSIX.1-2008 makes no representations concerning the format of the printed  
93639 output, except that it must be “human-readable” and “non-volatile”. Thus, writing by default to  
93640 a disk or tape drive or a display terminal would not qualify. (Such destinations are not  
93641 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)
- 93642 This volume of POSIX.1-2008 is worded such that a “print job” consisting of multiple input files  
93643 possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with  
93644 another, but there is no statement that all the files or copies have to print out together.
- 93645 The `-c` option may imply a spooling operation, but this is not required. The utility can be  
93646 implemented to wait until the printer is ready and then wait until it is finished. Because of that,  
93647 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).
- 93648 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel  
93649 or find the status of a request using utilities not defined in this volume of POSIX.1-2008.
- 93650 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,  
93651 they used different names for the environment variable specifying the destination printer. Since  
93652 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence  
93653 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one  
93654 or the other environment variable, the `lp` utility is required to recognize both. If this was not  
93655 done, many applications would send output to unexpected output devices when users moved  
93656 from system to system.
- 93657 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests  
93658 have proposed additional options or operands or both that added functionality. The requests  
93659 included:
- 93660 • Wording *requiring* the output to be “hardcopy”
  - 93661 • A requirement for multiple printers
  - 93662 • Options for supporting various page-description languages
- 93663 Given that a compliant system is not required to even have a printer, placing further restrictions  
93664 upon the behavior of the printer is not useful. Since hardcopy format is so application-  
93665 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that  
93666 should be required on all compliant systems.
- 93667 The term *unspecified* is used in this section in lieu of *implementation-defined* as most known  
93668 implementations would not be able to make definitive statements in their conformance  
93669 documents; the existence and usage of printers is very dependent on how the system  
93670 administrator configures each individual system.
- 93671 Since the default destination, device type, queuing mechanisms, and acceptable forms of input  
93672 are all unspecified, usage guidelines for what a conforming application can do are as follows:
- 93673 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and  
93674 the files can be safely deleted or modified.
  - 93675 • Limit output to text files of reasonable line lengths and printable characters and include no  
93676 device-specific formatting information, such as a page description language. The meaning  
93677 of “reasonable” in this context can only be answered as a quality-of-implementation issue,  
93678 but it should be apparent from historical usage patterns in the industry and the locale. The  
93679 `pr` and `fold` utilities can be used to achieve reasonable formatting for the default page size  
93680 of the implementation.

93681 Alternatively, the application can arrange its installation in such a way that it requires the system  
 93682 administrator or operator to provide the appropriate information on *lp* options and environment  
 93683 variable values.

93684 At a minimum, having this utility in this volume of POSIX.1-2008 tells the industry that  
 93685 conforming applications require a means to print output and provides at least a command name  
 93686 and *LPDEST* routing mechanism that can be used for discussions between vendors, application  
 93687 developers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent  
 93688 of the standard developers, even if they cannot mandate that all systems (such as laptops) have  
 93689 printers.

93690 This volume of POSIX.1-2008 does not specify what the ownership of the process performing the  
 93691 writing to the output device may be. If *-c* is not used, it is unspecified whether the process  
 93692 performing the writing to the output device has permission to read *file* if there are any  
 93693 restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the  
 93694 results of deleting *file* before it is printed are unspecified.

#### 93695 FUTURE DIRECTIONS

93696 None.

#### 93697 SEE ALSO

93698 *mailx*

93699 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

#### 93700 CHANGE HISTORY

93701 First released in Issue 2.

#### 93702 Issue 6

93703 The following new requirements on POSIX implementations derive from alignment with the  
 93704 Single UNIX Specification:

- 93705 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal  
 93706 generation of a banner page is added.
- 93707 • In the OPTIONS section:
  - 93708 — The *-d dest* description is expanded, but references to *lpstat* are removed.
  - 93709 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- 93710 • In the ENVIRONMENT VARIABLES section, *LC\_TIME* may now affect the execution.
- 93711 • The STDOUT section is added.

93712 The normative text is reworded to avoid use of the term “must” for application requirements.

93713 The TZ entry is added to the ENVIRONMENT VARIABLES section.

#### 93714 Issue 7

93715 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**ls**93716 **NAME**93717 **ls** — list directory contents93718 **SYNOPSIS**93719 xSI **ls** [-ACFRSacdfiklmnpqrstux1] [-H|-L] [-go] [*file...*]93720 **DESCRIPTION**

93721 For each operand that names a file of a type other than directory or symbolic link to a directory,  
 93722 *ls* shall write the name of the file as well as any requested, associated information. For each  
 93723 operand that names a file of type directory, *ls* shall write the names of files contained within the  
 93724 directory as well as any requested, associated information. Filenames beginning with a <period>  
 93725 ('.') and any associated information shall not be written out unless explicitly referenced, the  
 93726 **-A** or **-a** option is supplied, or an implementation-defined condition causes them to be written.  
 93727 If one or more of the **-d**, **-F**, or **-l** options are specified, and neither the **-H** nor the **-L** option is  
 93728 specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the  
 93729 name of the file as well as any requested, associated information. If none of the **-d**, **-F**, or **-l**  
 93730 options are specified, or the **-H** or **-L** options are specified, for each operand that names a file of  
 93731 type symbolic link to a directory, *ls* shall write the names of files contained within the directory  
 93732 as well as any requested, associated information. In each case where the names of files contained  
 93733 within a directory are written, if the directory contains any symbolic links then *ls* shall evaluate  
 93734 the file information and file type to be those of the symbolic link itself, unless the **-L** option is  
 93735 specified.

93736 If no operands are specified, *ls* shall behave as if a single operand of dot ('.') had been  
 93737 specified. If more than one operand is specified, *ls* shall write non-directory operands first; it  
 93738 shall sort directory and non-directory operands separately according to the collating sequence in  
 93739 the current locale.

93740 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 93741 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic  
 93742 message to standard error and shall either recover its position in the hierarchy or terminate.

93743 **OPTIONS**93744 The *ls* utility shall conform to XBD [Section 12.2](#) (on page 215).

93745 The following options shall be supported:

93746 **-A** Write out all directory entries, including those whose names begin with a <period>  
 93747 ('.') but excluding the entries dot and dot-dot (if they exist).

93748 **-C** Write multi-text-column output with entries sorted down the columns, according  
 93749 to the collating sequence. The number of text columns and the column separator  
 93750 characters are unspecified, but should be adapted to the nature of the output  
 93751 device.

93752 **-F** Do not follow symbolic links named as operands unless the **-H** or **-L** options are  
 93753 specified. Write a <slash> ('/') immediately after each pathname that is a  
 93754 directory, an <asterisk> ('\*') after each that is executable, a <vertical-line> ('|')  
 93755 after each that is a FIFO, and an at-sign ('@') after each that is a symbolic link. For  
 93756 other file types, other symbols may be written.

93757 **-H** Evaluate the file information and file type for symbolic links specified on the  
 93758 command line to be those of the file referenced by the link, and not the link itself;  
 93759 however, *ls* shall write the name of the link itself and not the file referenced by the  
 93760 link.

|       |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|-----|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 93761 |     | <b>-L</b> | Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link. When <b>-L</b> is used with <b>-l</b> , write the contents of symbolic links in the long format (see the STDOUT section). |
| 93762 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93763 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93764 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93765 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93766 |     | <b>-R</b> | Recursively list subdirectories encountered. When a symbolic link to a directory is encountered, the directory shall not be recursively listed unless the <b>-L</b> option is specified.                                                                                                                                                                                                                                                         |
| 93767 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93768 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93769 |     | <b>-S</b> | Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).                                                                                                                                                                                                                                                                                            |
| 93770 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93771 |     | <b>-a</b> | Write out all directory entries, including those whose names begin with a <period> ('.').                                                                                                                                                                                                                                                                                                                                                        |
| 93772 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93773 |     | <b>-c</b> | Use time of last modification of the file status information (see <sys/stat.h> in the System Interfaces volume of POSIX.1-2008) instead of last modification of the file itself for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                                                                                                              |
| 93774 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93775 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93776 |     | <b>-d</b> | Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Do not treat directories differently than other types of files. The use of <b>-d</b> with <b>-R</b> produces unspecified results.                                                                                                                                                                                                        |
| 93777 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93778 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93779 |     | <b>-f</b> | List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn off <b>-l</b> , <b>-t</b> , <b>-S</b> , <b>-s</b> , and <b>-r</b> , and shall turn on <b>-a</b> .                                                                                                                                                                               |
| 93780 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93781 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93782 | XSI | <b>-g</b> | The same as <b>-l</b> , except that the owner shall not be written.                                                                                                                                                                                                                                                                                                                                                                              |
| 93783 |     | <b>-i</b> | For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces volume of POSIX.1-2008).                                                                                                                                                                                                                                                                                                                          |
| 93784 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93785 |     | <b>-k</b> | Set the block size for the <b>-s</b> option and the per-directory block count written for the <b>-l</b> , <b>-n</b> , <b>-s</b> , <b>-g</b> , and <b>-o</b> options (see the STDOUT section) to 1 024 bytes.                                                                                                                                                                                                                                     |
| 93786 | XSI |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93787 |     | <b>-l</b> | (The letter ell.) Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Write out in long format (see the STDOUT section). When <b>-l</b> (ell) is specified, <b>-l</b> (one) shall be assumed.                                                                                                                                                                                                |
| 93788 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93789 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93790 |     | <b>-m</b> | Stream output format; list files across the page, separated by <comma> characters.                                                                                                                                                                                                                                                                                                                                                               |
| 93791 |     | <b>-n</b> | The same as <b>-l</b> , except that the owner's UID and GID numbers shall be written, rather than the associated character strings.                                                                                                                                                                                                                                                                                                              |
| 93792 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93793 | XSI | <b>-o</b> | The same as <b>-l</b> , except that the group shall not be written.                                                                                                                                                                                                                                                                                                                                                                              |
| 93794 |     | <b>-p</b> | Write a <slash> ('/') after each filename if that file is a directory.                                                                                                                                                                                                                                                                                                                                                                           |
| 93795 |     | <b>-q</b> | Force each instance of non-printable filename characters and <tab> characters to be written as the <question-mark> ('?') character. Implementations may provide this option by default if the output is to a terminal device.                                                                                                                                                                                                                    |
| 93796 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93797 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93798 |     | <b>-r</b> | Reverse the order of the sort to get reverse collating sequence oldest first, or smallest file size first depending on the other options given.                                                                                                                                                                                                                                                                                                  |
| 93799 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93800 |     | <b>-s</b> | Indicate the total number of file system blocks consumed by each file displayed. If the <b>-k</b> option is also specified, the block size shall be 1 024 bytes; otherwise, the block size is implementation-defined.                                                                                                                                                                                                                            |
| 93801 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 93802 |     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

- 93803        **-t**            Sort with the primary key being time modified (most recently modified first) and  
 93804                    the secondary key being filename in the collating sequence. For a symbolic link,  
 93805                    the time used as the sort key is that of the symbolic link itself, unless *ls* is  
 93806                    evaluating its file information to be that of the file referenced by the link (see the  
 93807                    **-H** and **-L** options).
- 93808        **-u**            Use time of last access (see **<sys/stat.h>**) instead of last modification of the file for  
 93809                    sorting (**-t**) or writing (**-l**).
- 93810        **-x**            The same as **-C**, except that the multi-text-column output is produced with entries  
 93811                    sorted across, rather than down, the columns.
- 93812        **-1**            (The numeric digit one.) Force output to be one entry per line.
- 93813                    Specifying more than one of the options in the following mutually-exclusive pairs shall not be  
 93814                    considered an error: **-C** and **-l** (ell), **-m** and **-l** (ell), **-x** and **-l** (ell), **-C** and **-1** (one), **-H** and **-L**,  
 93815                    **-c** and **-u**, **-t** and **-S**. The last option specified in each pair shall determine the output format.
- 93816        **OPERANDS**
- 93817                    The following operand shall be supported:
- 93818        *file*            A pathname of a file to be written. If the file specified is not found, a diagnostic  
 93819                    message shall be output on standard error.
- 93820        **STDIN**
- 93821                    Not used.
- 93822        **INPUT FILES**
- 93823                    None.
- 93824        **ENVIRONMENT VARIABLES**
- 93825                    The following environment variables shall affect the execution of *ls*:
- 93826        **COLUMNS**    Determine the user's preferred column position width for writing multiple text-  
 93827                    column output. If this variable contains a string representing a decimal integer, the  
 93828                    *ls* utility shall calculate how many pathname text columns to write (see **-C**) based  
 93829                    on the width provided. If **COLUMNS** is not set or invalid, an implementation-  
 93830                    defined number of column positions shall be assumed, based on the  
 93831                    implementation's knowledge of the output device. The column width chosen to  
 93832                    write the names of files in any given directory shall be constant. Filenames shall  
 93833                    not be truncated to fit into the multiple text-column output.
- 93834        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 93835                    (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 93836                    variables used to determine the values of locale categories.)
- 93837        **LC\_ALL**            If set to a non-empty string value, override the values of all the other  
 93838                    internationalization variables.
- 93839        **LC\_COLLATE**
- 93840                    Determine the locale for character collation information in determining the  
 93841                    pathname collation sequence.
- 93842        **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as  
 93843                    characters (for example, single-byte as opposed to multi-byte characters in  
 93844                    arguments) and which characters are defined as printable (character class **print**).

- 93845 *LC\_MESSAGES*
- 93846 Determine the locale that should be used to affect the format and contents of
- 93847 diagnostic messages written to standard error.
- 93848 *LC\_TIME* Determine the format and contents for date and time strings written by *ls*.
- 93849 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 93850 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or
- 93851 null, an unspecified default timezone shall be used.
- 93852 **ASYNCHRONOUS EVENTS**
- 93853 Default.
- 93854 **STDOUT**
- 93855 The default format shall be to list one entry per line to standard output; the exceptions are to
- 93856 terminals or when one of the *-C*, *-m*, or *-x* options is specified. If the output is to a terminal, the
- 93857 format is implementation-defined.
- 93858 When *-m* is specified, the format used shall be:
- 93859 "%s, %s, ...\n", <filename1>, <filename2>
- 93860 where the largest number of filenames shall be written without exceeding the length of the line.
- 93861 If the *-i* option is specified, the file's file serial number (see <sys/stat.h>) shall be written in the
- 93862 following format before any other output for the corresponding entry:
- 93863 %u ", <file serial number>
- 93864 If the *-l* option is specified without *-L*, the following information shall be written:
- 93865 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
93866 <owner name>, <group name>, <number of bytes in the file>,  
93867 <date and time>, <pathname>
- 93868 If the file is a symbolic link, this information shall be about the link itself and the <pathname>
- 93869 field shall be of the form:
- 93870 "%s -> %s", <pathname of link>, <contents of link>
- 93871 If both *-l* and *-L* are specified, the following information shall be written:
- 93872 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
93873 <owner name>, <group name>, <number of bytes in the file>,  
93874 <date and time>, <pathname of link>
- 93875 where all fields except <pathname of link> shall be for the file resolved from the symbolic link.
- 93876 XSI The *-n*, *-g*, and *-o* options use the same format as *-l*, but with omitted items and their
- 93877 associated <blank> characters. See the OPTIONS section.
- 93878 In both the preceding *-l* forms, if <owner name> or <group name> cannot be determined, or if *-n*
- 93879 is given, they shall be replaced with their associated numeric values using the format %u.
- 93880 The <date and time> field shall contain the appropriate date and timestamp of when the file was
- 93881 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
- 93882 *date* command:
- 93883 date "+%b %e %H:%M"
- 93884 if the file has been modified in the last six months, or:

93885           date "+%b %e %Y"

93886           (where two <space> characters are used between %e and %Y) if the file has not been modified in  
93887           the last six months or if the modification date is in the future, except that, in both cases, the final  
93888           <newline> produced by *date* shall not be included and the output shall be as if the *date*  
93889           command were executed at the time of the last modification date of the file rather than the  
93890           current time. When the *LC\_TIME* locale category is not set to the POSIX locale, a different format  
93891           and order of presentation of this field may be used.

93892           If the file is a character special or block special file, the size of the file may be replaced with  
93893           implementation-defined information associated with the device in question.

93894           If the pathname was specified as a *file* operand, it shall be written as specified.

93895 XSI        The file mode written under the *-l*, *-n*, *-g*, and *-o* options shall consist of the following format:

93896           "%c%s%s%s", <entry type>, <owner permissions>,  
93897                    <group permissions>, <other permissions>,  
93898                    <optional alternate access method flag>

93899           The <optional alternate access method flag> shall be the empty string if there is no alternate or  
93900           additional access control method associated with the file; otherwise, it shall be a string  
93901           containing a single printable character that is not a <blank>.

93902           The <entry type> character shall describe the type of file, as follows:

93903           d        Directory.  
93904           b        Block special file.  
93905           c        Character special file.  
93906           l (ell) Symbolic link.  
93907           p        FIFO.  
93908           -        Regular file.

93909           Implementations may add other characters to this list to represent other implementation-defined  
93910           file types.

93911           The next three fields shall be three characters each:

93912           <owner permissions>  
93913                Permissions for the file owner class (see XBD Section 4.4, on page 108).  
93914           <group permissions>  
93915                Permissions for the file group class.  
93916           <other permissions>  
93917                Permissions for the file other class.

93918           Each field shall have three character positions:

- 93919           1. If 'r', the file is readable; if '-', the file is not readable.  
93920           2. If 'w', the file is writable; if '-', the file is not writable.  
93921           3. The first of the following that applies:
- 93922           S    If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in  
93923                <group permissions>, the file is not executable and set-group-ID mode is set.

- 93924 s If in *<owner permissions>*, the file is executable and set-user-ID mode is set. If in  
93925 *<group permissions>*, the file is executable and set-group-ID mode is set.
- 93926 XSI T If in *<other permissions>* and the file is a directory, search permission is not granted to  
93927 others, and the restricted deletion flag is set.
- 93928 XSI t If in *<other permissions>* and the file is a directory, search permission is granted to  
93929 others, and the restricted deletion flag is set.
- 93930 x The file is executable or the directory is searchable.
- 93931 – None of the attributes of 'S', 's', 'T', 't', or 'x' applies.
- 93932 Implementations may add other characters to this list for the third character position.  
93933 Such additions shall, however, be written in lowercase if the file is executable or  
93934 searchable, and in uppercase if it is not.
- 93935 XSI If any of the **-l**, **-n**, **-s**, **-g**, or **-o** options is specified, each list of files within the directory shall be  
93936 preceded by a status line indicating the number of file system blocks occupied by files in the  
93937 directory in 512-byte units if the **-k** option is not specified, or 1 024-byte units if the **-k** option is  
93938 specified, rounded up to the next integral number of units, if necessary. In the POSIX locale, the  
93939 format shall be:
- 93940 "total %u\n", *<number of units in the directory>*
- 93941 If more than one directory, or a combination of non-directory files and directories are written,  
93942 either as a result of specifying multiple operands, or the **-R** option, each list of files within a  
93943 directory shall be preceded by:
- 93944 "\n%s:\n", *<directory name>*
- 93945 If this string is the first thing to be written, the first *<newline>* shall not be written. This output  
93946 shall precede the number of units in the directory.
- 93947 If the **-s** option is given, each file shall be written with the number of blocks used by the file.  
93948 XSI Along with **-C**, **-l**, **-m**, or **-x**, the number and a *<space>* shall precede the filename; with **-l**, **-n**,  
93949 **-g**, or **-o**, they shall precede each line describing a file.
- 93950 **STDERR**
- 93951 The standard error shall be used only for diagnostic messages.
- 93952 **OUTPUT FILES**
- 93953 None.
- 93954 **EXTENDED DESCRIPTION**
- 93955 None.
- 93956 **EXIT STATUS**
- 93957 The following exit values shall be returned:
- 93958 0 Successful completion.
- 93959 >0 An error occurred.
- 93960 **CONSEQUENCES OF ERRORS**
- 93961 Default.

## 93962 APPLICATION USAGE

93963 Many implementations use the <equals-sign> ('=') to denote sockets bound to the file system  
 93964 for the **-F** option. Similarly, many historical implementations use the 's' character to denote  
 93965 sockets as the entry type characters for the **-l** option.

93966 It is difficult for an application to use every part of the file modes field of *ls -l* in a portable  
 93967 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as  
 93968 implementations may have extensions. Applications can use this field to pass directly to a user  
 93969 printout or prompt, but actions based on its contents should generally be deferred, instead, to  
 93970 the *test* utility.

93971 The output of *ls* (with the **-l** and related options) contains information that logically could be  
 93972 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this  
 93973 information is presented in a format that cannot be used directly by those utilities or be easily  
 93974 translated into a format that can be used. A character has been added to the end of the  
 93975 permissions string so that applications at least have an indication that they may be working in  
 93976 an area they do not understand instead of assuming that they can translate the permissions  
 93977 string into something that can be used. Future versions or related documents may define one or  
 93978 more specific characters to be used based on different standard additional or alternative access  
 93979 control mechanisms.

93980 As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one  
 93981 of the long listing formats must be used carefully on systems where filenames can contain  
 93982 embedded white space. Systems and system administrators should institute policies and user  
 93983 training to limit the use of such filenames.

93984 The number of disk blocks occupied by the file that it reports varies depending on underlying  
 93985 file system type, block size units reported, and the method of calculating the number of blocks.  
 93986 On some file system types, the number is the actual number of blocks occupied by the file  
 93987 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the  
 93988 file size (usually making an allowance for indirect blocks, but ignoring holes).

## 93989 EXAMPLES

93990 An example of a small directory tree being fully listed with *ls -laRF a* in the POSIX locale:

```
93991 total 11
93992 drwxr-xr-x  3 fox      prog          64 Jul  4 12:07 ./
93993 drwxrwxrwx  4 fox      prog        3264 Jul  4 12:09 ../
93994 drwxr-xr-x  2 fox      prog          48 Jul  4 12:07 b/
93995 -rwxr--r--  1 fox      prog         572 Jul  4 12:07 foo*

93996 a/b:
93997 total 4
93998 drwxr-xr-x  2 fox      prog          48 Jul  4 12:07 ./
93999 drwxr-xr-x  3 fox      prog          64 Jul  4 12:07 ../
94000 -rw-r--r--  1 fox      prog         700 Jul  4 12:07 bar
```

## 94001 RATIONALE

94002 Some historical implementations of the *ls* utility show all entries in a directory except dot and  
 94003 dot-dot when a superuser invokes *ls* without specifying the **-a** option. When "normal" users  
 94004 invoke *ls* without specifying **-a**, they should not see information about any files with names  
 94005 beginning with a <period> unless they were named as *file* operands.

94006 Implementations are expected to traverse arbitrary depths when processing the **-R** option. The  
 94007 only limitation on depth should be based on running out of physical storage for keeping track of  
 94008 untraversed directories.

94009 The `-1` (one) option was historically found in BSD and BSD-derived implementations only. It is  
 94010 required in this volume of POSIX.1-2008 so that conforming applications might ensure that  
 94011 output is one entry per line, even if the output is to a terminal.

94012 The `-S` option was added in Issue 7, but had been provided by several implementations for  
 94013 many years. The description given in the standard documents historic practice, but does not  
 94014 match much of the documentation that described its behavior. Historical documentation  
 94015 typically described it as something like:

94016 `-S` Sort by size (largest size first) instead of by name. Special character devices (listed  
 94017 last) are sorted by name.

94018 even though the file type was never considered when sorting the output. Character special files  
 94019 do typically sort close to the end of the list because their file size on most implementations is  
 94020 zero. But they are sorted alphabetically with any other files that happen to have the same file  
 94021 size (zero), not sorted separately and added to the end.

94022 Generally, this volume of POSIX.1-2008 is silent about what happens when options are given  
 94023 multiple times. In the cases of `-C`, `-l`, and `-1`, however, it does specify the results of these  
 94024 overlapping options. Since `ls` is one of the most aliased commands, it is important that the  
 94025 implementation perform intuitively. For example, if the alias were:

```
94026 alias ls="ls -C"
```

94027 and the user typed `ls -1`, single-text-column output should result, not an error.

94028 Earlier versions of this standard did not describe the BSD `-A` option (like `-a`, but dot and dot-dot  
 94029 are not written out). It has been added due to widespread implementation.

94030 Implementations may make `-q` the default for terminals to prevent trojan horse attacks on  
 94031 terminals with special escape sequences. This is not required because:

- 94032 • Some control characters may be useful on some terminals; for example, a system might  
 94033 write them as `"\001"` or `"^A"`.
- 94034 • Special behavior for terminals is not relevant to applications portability.

94035 An early proposal specified that the *<optional alternate access method flag>* had to be `'+'` if there  
 94036 was an alternate access method used on the file or `<space>` if there was not. This was changed to  
 94037 be `<space>` if there is not and a single printable character if there is. This was done for three  
 94038 reasons:

- 94039 1. There are historical implementations using characters other than `'+'`.
- 94040 2. There are implementations that vary this character used in that position to distinguish  
 94041 between various alternate access methods in use.
- 94042 3. The standard developers did not want to preclude future specifications that might need a  
 94043 way to specify more than one alternate access method.

94044 Nonetheless, implementations providing a single alternate access method are encouraged to use  
 94045 `'+'`.

94046 Earlier versions of this standard did not have the `-k` option, which meant that the `-s` option  
 94047 could not be used portably as its block size was implementation-defined, and the units used to  
 94048 specify the number of blocks occupied by files in a directory in an `ls -l` listing were fixed as  
 94049 512-byte units. The `-k` option has been added to provide a way for the `-s` option to be used  
 94050 portably, and for consistency it also changes the aforementioned units from 512-byte to  
 94051 1024-byte.

- 94052 The *<date and time>* field in the `-l` format is specified only for the POSIX locale. As noted, the  
 94053 format can be different in other locales. No mechanism for defining this is present in this volume  
 94054 of POSIX.1-2008, as the appropriate vehicle is a messaging system; that is, the format should be  
 94055 specified as a “message”.
- 94056 **FUTURE DIRECTIONS**
- 94057 None.
- 94058 **SEE ALSO**
- 94059 *chmod*, *find*
- 94060 XBD Section 4.4 (on page 108), Chapter 8 (on page 173), Section 12.2 (on page 215), `<sys/stat.h>`  
 94061 XSH *fstatat()*
- 94062 **CHANGE HISTORY**
- 94063 First released in Issue 2.
- 94064 **Issue 5**
- 94065 A second FUTURE DIRECTION is added.
- 94066 **Issue 6**
- 94067 The following new requirements on POSIX implementations derive from alignment with the  
 94068 Single UNIX Specification:
- 94069 • In the `-F` option, other symbols are allowed for other file types.
- 94070 Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.
- 94071 The Open Group Base Resolution bwg2001-010 is applied, adding the `T` and `t` fields as part of  
 94072 the XSI option.
- 94073 **Issue 7**
- 94074 Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access  
 94075 method flag in the `STDOUT` section.
- 94076 Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the `DESCRIPTION` and the  
 94077 definition of the `-R` option.
- 94078 Austin Group Interpretation 1003.1-2001 #129 is applied, clarifying the behavior of *ls* when no  
 94079 operands are specified.
- 94080 Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the `-H`  
 94081 option.
- 94082 SD5-XCU-ERN-50 is applied, adding the `-A` option.
- 94083 SD5-XCU-ERN-97 is applied, updating the `SYNOPSIS`.
- 94084 The `-S` option is added from The Open Group Technical Standard, 2006, Extended API Set  
 94085 Part 1.
- 94086 The `-f`, `-m`, `-n`, `-p`, `-s`, and `-x` options are moved from the XSI option to the Base.
- 94087 The description of the `-f`, `-s`, and `-t` options are revised and the `-k` option is added.

94088 **NAME**94089 `m4` — macro processor94090 **SYNOPSIS**94091 `m4 [-s] [-D name[=val]]... [-U name]... file...`94092 **DESCRIPTION**94093 The *m4* utility is a macro processor that shall read one or more text files, process them according  
94094 to their included macro statements, and write the results to standard output.94095 **OPTIONS**94096 The *m4* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the `-D`  
94097 and `-U` options shall be significant, and options can be interspersed with operands.

94098 The following options shall be supported:

94099 `-s` Enable line synchronization output for the *c99* preprocessor phase (that is, `#line`  
94100 directives).94101 `-D name[=val]`94102 Define *name* to *val* or to null if *=val* is omitted.94103 `-U name` Undefine *name*.94104 **OPERANDS**

94105 The following operand shall be supported:

94106 *file* A pathname of a text file to be processed. If no *file* is given, or if it is `'-'`, the  
94107 standard input shall be read.94108 **STDIN**94109 The standard input shall be a text file that is used if no *file* operand is given, or if it is `'-'`.94110 **INPUT FILES**94111 The input file named by the *file* operand shall be a text file.94112 **ENVIRONMENT VARIABLES**94113 The following environment variables shall affect the execution of *m4*:94114 *LANG* Provide a default value for the internationalization variables that are unset or null.  
94115 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
94116 variables used to determine the values of locale categories.)94117 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
94118 internationalization variables.94119 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
94120 characters (for example, single-byte as opposed to multi-byte characters in  
94121 arguments and input files).94122 *LC\_MESSAGES*94123 Determine the locale that should be used to affect the format and contents of  
94124 diagnostic messages written to standard error.94125 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.94126 **ASYNCHRONOUS EVENTS**

94127 Default.

94128 **STDOUT**

94129 The standard output shall be the same as the input files, after being processed for macro  
94130 expansion.

94131 **STDERR**

94132 The standard error shall be used to display strings with the **errprint** macro, macro tracing  
94133 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for  
94134 diagnostic messages.

94135 **OUTPUT FILES**

94136 None.

94137 **EXTENDED DESCRIPTION**

94138 The *m4* utility shall compare each token from the input against the set of built-in and user-  
94139 defined macros. If the token matches the name of a macro, then the token shall be replaced by  
94140 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of  
94141 the token matches the name of a macro, it shall be written to standard output. Macros may have  
94142 arguments, in which case the arguments shall be substituted into the defining text before it is  
94143 rescanned.

94144 Macro calls have the form:

94145 *name*(*arg1*, *arg2*, ..., *argn*)

94146 Macro names shall consist of letters, digits, and underscores, where the first character is not a  
94147 digit. Tokens not of this form shall not be treated as macros.

94148 The application shall ensure that the <left-parenthesis> immediately follows the name of the  
94149 macro. If a token matching the name of a macro is not followed by a <left-parenthesis>, it is  
94150 handled as a use of that macro without arguments.

94151 If a macro name is followed by a <left-parenthesis>, its arguments are the <comma>-separated  
94152 tokens between the <left-parenthesis> and the matching <right-parenthesis>. Unquoted white-  
94153 space characters preceding each argument shall be ignored. All other characters, including  
94154 trailing white-space characters, are retained. <comma> characters enclosed between <left-  
94155 parenthesis> and <right-parenthesis> characters do not delimit arguments.

94156 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be  
94157 replaced by the first argument. Systems shall support at least nine arguments; only the first nine  
94158 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with  
94159 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The  
94160 string "\$\*" is replaced by a list of all of the arguments, separated by <comma> characters. The  
94161 string "\$@" is replaced by a list of all of the arguments separated by <comma> characters, and  
94162 each argument is quoted using the current left and right quoting strings. The string "\${"  
94163 produces unspecified behavior.

94164 If fewer arguments are supplied than are in the macro definition, the omitted arguments are  
94165 taken to be null. It is not an error if more arguments are supplied than are in the macro  
94166 definition.

94167 No special meaning is given to any characters enclosed between matching left and right quoting  
94168 strings, but the quoting strings are themselves discarded. By default, the left quoting string  
94169 consists of a grave accent (backquote) and the right quoting string consists of an acute accent  
94170 (single-quote); see also the **changequote** macro.

94171 Comments are written but not scanned for matching macro names; by default, the begin-  
94172 comment string consists of the <number-sign> character and the end-comment string consists of  
94173 a <newline>. See also the **changecom** and **dnl** macros.

- 94174 The *m4* utility shall make available the following built-in macros. They can be redefined, but  
 94175 once this is done the original meaning is lost. Their values shall be null unless otherwise stated.  
 94176 In the descriptions below, the term *defining text* refers to the value of the macro: the second  
 94177 argument to the **define** macro, among other things. Except for the first argument to the **eval**  
 94178 macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The  
 94179 string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval**  
 94180 built-in macros shall be in the form of a decimal-constant as defined in the C language.
- 94181 **changecom** The **changecom** macro shall set the begin-comment and end-comment strings.  
 94182 With no arguments, the comment mechanism shall be disabled. With a single non-  
 94183 null argument, that argument shall become the begin-comment and the <newline>  
 94184 shall become the end-comment string. With two non-null arguments, the first  
 94185 argument shall become the begin-comment string and the second argument shall  
 94186 become the end-comment string. The behavior is unspecified if either argument is  
 94187 provided but null. Systems shall support comment strings of at least five  
 94188 characters.
- 94189 **changequote** The **changequote** macro shall set the begin-quote and end-quote strings. With no  
 94190 arguments, the quote strings shall be set to the default values (that is, `'). The  
 94191 behavior is unspecified if there is a single argument or either argument is null.  
 94192 With two non-null arguments, the first argument shall become the begin-quote  
 94193 string and the second argument shall become the end-quote string. Systems shall  
 94194 support quote strings of at least five characters.
- 94195 **decr** The defining text of the **decr** macro shall be its first argument decremented by 1. It  
 94196 shall be an error to specify an argument containing any non-numeric characters.  
 94197 The behavior is unspecified if **decr** is not immediately followed by a <left-  
 94198 parenthesis>.
- 94199 **define** The second argument shall become the defining text of the macro whose name is  
 94200 the first argument. It is unspecified whether the **define** macro deletes all prior  
 94201 definitions of the macro named by its first argument or preserves all but the  
 94202 current definition of the macro. The behavior is unspecified if **define** is not  
 94203 immediately followed by a <left-parenthesis>.
- 94204 **defn** The defining text of the **defn** macro shall be the quoted definition (using the  
 94205 current quoting strings) of its arguments. The behavior is unspecified if **defn** is not  
 94206 immediately followed by a <left-parenthesis>.
- 94207 **divert** The *m4* utility maintains nine temporary buffers, numbered 1 to 9, inclusive.  
 94208 When the last of the input has been processed, any output that has been placed in  
 94209 these buffers shall be written to standard output in buffer-numerical order. The  
 94210 **divert** macro shall divert future output to the buffer specified by its argument.  
 94211 Specifying no argument or an argument of 0 shall resume the normal output  
 94212 process. Output diverted to a stream with a negative number shall be discarded.  
 94213 Behavior is implementation-defined if a stream number larger than 9 is specified. It  
 94214 shall be an error to specify an argument containing any non-numeric characters.
- 94215 **divnum** The defining text of the **divnum** macro shall be the number of the current output  
 94216 stream as a string.
- 94217 **dnl** The **dnl** macro shall cause *m4* to discard all input characters up to and including  
 94218 the next <newline>.

|       |                 |                                                                                                                                                                                                                                                                                                                   |
|-------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 94219 | <b>dumpdef</b>  | The <b>dumpdef</b> macro shall write the defined text to standard error for each of the macros specified as arguments, or, if no arguments are specified, for all macros.                                                                                                                                         |
| 94220 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94221 | <b>errprint</b> | The <b>errprint</b> macro shall write its arguments to standard error. The behavior is unspecified if <b>errprint</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                        |
| 94222 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94223 | <b>eval</b>     | The <b>eval</b> macro shall evaluate its first argument as an arithmetic expression, using signed integer arithmetic with at least 32-bit precision. At least the following C language operators shall be supported, with precedence, associativity, and behavior as described in Section 1.1.2.1 (on page 2283): |
| 94224 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94225 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94226 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94227 |                 | ( )                                                                                                                                                                                                                                                                                                               |
| 94228 |                 | unary +                                                                                                                                                                                                                                                                                                           |
| 94229 |                 | unary -                                                                                                                                                                                                                                                                                                           |
| 94230 |                 | ~                                                                                                                                                                                                                                                                                                                 |
| 94231 |                 | !                                                                                                                                                                                                                                                                                                                 |
| 94232 |                 | binary *                                                                                                                                                                                                                                                                                                          |
| 94233 |                 | /                                                                                                                                                                                                                                                                                                                 |
| 94234 |                 | %                                                                                                                                                                                                                                                                                                                 |
| 94235 |                 | binary +                                                                                                                                                                                                                                                                                                          |
| 94236 |                 | binary -                                                                                                                                                                                                                                                                                                          |
| 94237 |                 | <<                                                                                                                                                                                                                                                                                                                |
| 94238 |                 | >>                                                                                                                                                                                                                                                                                                                |
| 94239 |                 | <                                                                                                                                                                                                                                                                                                                 |
| 94240 |                 | <=                                                                                                                                                                                                                                                                                                                |
| 94241 |                 | >                                                                                                                                                                                                                                                                                                                 |
| 94242 |                 | >=                                                                                                                                                                                                                                                                                                                |
| 94243 |                 | ==                                                                                                                                                                                                                                                                                                                |
| 94244 |                 | !=                                                                                                                                                                                                                                                                                                                |
| 94245 |                 | binary &                                                                                                                                                                                                                                                                                                          |
| 94246 |                 | ^                                                                                                                                                                                                                                                                                                                 |
| 94247 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94248 |                 | &&                                                                                                                                                                                                                                                                                                                |
| 94249 |                 |                                                                                                                                                                                                                                                                                                                   |
| 94250 |                 | Systems shall support octal and hexadecimal numbers as in the ISO C standard.                                                                                                                                                                                                                                     |
| 94251 |                 | The second argument, if specified, shall set the radix for the result; if the argument                                                                                                                                                                                                                            |
| 94252 |                 | is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls                                                                                                                                                                                                                            |
| 94253 |                 | outside the range 2 to 36, inclusive. The third argument, if specified, sets the                                                                                                                                                                                                                                  |
| 94254 |                 | minimum number of digits in the result. Behavior is unspecified if the third                                                                                                                                                                                                                                      |
| 94255 |                 | argument is less than zero. It shall be an error to specify the second or third                                                                                                                                                                                                                                   |
| 94256 |                 | argument containing any non-numeric characters. The behavior is unspecified if                                                                                                                                                                                                                                    |
| 94257 |                 | <b>eval</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                  |
| 94258 | <b>ifdef</b>    | If the first argument to the <b>ifdef</b> macro is defined, the defining text shall be the                                                                                                                                                                                                                        |
| 94259 |                 | second argument. Otherwise, the defining text shall be the third argument, if                                                                                                                                                                                                                                     |
| 94260 |                 | specified, or the null string, if not. The behavior is unspecified if <b>ifdef</b> is not                                                                                                                                                                                                                         |
| 94261 |                 | immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                     |
| 94262 | <b>ifelse</b>   | The <b>ifelse</b> macro takes three or more arguments. If the first two arguments                                                                                                                                                                                                                                 |
| 94263 |                 | compare as equal strings (after macro expansion of both arguments), the defining                                                                                                                                                                                                                                  |
| 94264 |                 | text shall be the third argument. If the first two arguments do not compare as equal                                                                                                                                                                                                                              |
| 94265 |                 | strings and there are three arguments, the defining text shall be null. If the first two                                                                                                                                                                                                                          |
| 94266 |                 | arguments do not compare as equal strings and there are four or five arguments,                                                                                                                                                                                                                                   |

|       |                    |                                                                                                 |
|-------|--------------------|-------------------------------------------------------------------------------------------------|
| 94267 |                    | the defining text shall be the fourth argument. If the first two arguments do not               |
| 94268 |                    | compare as equal strings and there are six or more arguments, the first three                   |
| 94269 |                    | arguments shall be discarded and processing shall restart with the remaining                    |
| 94270 |                    | arguments. The behavior is unspecified if <b>ifelse</b> is not immediately followed by a        |
| 94271 |                    | <left-parenthesis>.                                                                             |
| 94272 | <b>include</b>     | The defining text for the <b>include</b> macro shall be the contents of the file named by       |
| 94273 |                    | the first argument. It shall be an error if the file cannot be read. The behavior is            |
| 94274 |                    | unspecified if <b>include</b> is not immediately followed by a <left-parenthesis>.              |
| 94275 | <b>incr</b>        | The defining text of the <b>incr</b> macro shall be its first argument incremented by 1. It     |
| 94276 |                    | shall be an error to specify an argument containing any non-numeric characters.                 |
| 94277 |                    | The behavior is unspecified if <b>incr</b> is not immediately followed by a <left-              |
| 94278 |                    | parenthesis>.                                                                                   |
| 94279 | <b>index</b>       | The defining text of the <b>index</b> macro shall be the first character position (as a         |
| 94280 |                    | string) in the first argument where a string matching the second argument begins                |
| 94281 |                    | (zero origin), or -1 if the second argument does not occur. The behavior is                     |
| 94282 |                    | unspecified if <b>index</b> is not immediately followed by a <left-parenthesis>.                |
| 94283 | <b>len</b>         | The defining text of the <b>len</b> macro shall be the length (as a string) of the first        |
| 94284 |                    | argument. The behavior is unspecified if <b>len</b> is not immediately followed by a            |
| 94285 |                    | <left-parenthesis>.                                                                             |
| 94286 | <b>m4exit</b>      | Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The   |
| 94287 |                    | default is zero. It shall be an error to specify an argument containing any non-                |
| 94288 |                    | numeric characters.                                                                             |
| 94289 | <b>m4wrap</b>      | The first argument shall be processed when EOF is reached. If the <b>m4wrap</b> macro           |
| 94290 |                    | is used multiple times, the arguments specified shall be processed in the order in              |
| 94291 |                    | which the <b>m4wrap</b> macros were processed. The behavior is unspecified if <b>m4wrap</b>     |
| 94292 |                    | is not immediately followed by a <left-parenthesis>.                                            |
| 94293 | OB <b>maketemp</b> | The defining text shall be the first argument, with any trailing 'X' characters                 |
| 94294 |                    | replaced with the current process ID as a string. The behavior is unspecified if                |
| 94295 |                    | <b>maketemp</b> is not immediately followed by a <left-parenthesis>.                            |
| 94296 | <b>mkstemp</b>     | The first argument shall be taken as a template for creating an empty file, with                |
| 94297 |                    | trailing 'X' characters replaced with characters from the portable filename                     |
| 94298 |                    | character set. The behavior is unspecified if the first argument does not end in at             |
| 94299 |                    | least six 'X' characters. If a temporary file is successfully created, then the                 |
| 94300 |                    | defining text of the macro shall be the name of the new file. The user ID of the file           |
| 94301 |                    | shall be set to the effective user ID of the process. The group ID of the file shall be         |
| 94302 |                    | set to the group ID of the file's parent directory or to the effective group ID of the          |
| 94303 |                    | process. The file access permission bits are set such that only the owner can both              |
| 94304 |                    | read and write the file, regardless of the current <i>umask</i> of the process. If a file could |
| 94305 |                    | not be created, the defining text of the macro shall be the empty string. The                   |
| 94306 |                    | behavior is unspecified if <b>mkstemp</b> is not immediately followed by a <left-               |
| 94307 |                    | parenthesis>.                                                                                   |
| 94308 | <b>popdef</b>      | The <b>popdef</b> macro shall delete the current definition of its arguments, replacing         |
| 94309 |                    | that definition with the previous one. If there is no previous definition, the macro            |
| 94310 |                    | is undefined. The behavior is unspecified if <b>popdef</b> is not immediately followed by       |
| 94311 |                    | a <left-parenthesis>.                                                                           |

|       |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 94312 | <b>pushdef</b>  | The <b>pushdef</b> macro shall be equivalent to the <b>define</b> macro with the exception that it shall preserve any current definition for future retrieval using the <b>popdef</b> macro.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94313 |                 | The behavior is unspecified if <b>pushdef</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 94314 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94315 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94316 | <b>shift</b>    | The defining text for the <b>shift</b> macro shall be all of its arguments except for the first one. The behavior is unspecified if <b>shift</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 94317 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94318 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94319 | <b>sinclude</b> | The <b>sinclude</b> macro shall be equivalent to the <b>include</b> macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if <b>sinclude</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94320 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94321 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94322 | <b>substr</b>   | The defining text for the <b>substr</b> macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if <b>substr</b> is not immediately followed by a <left-parenthesis>. |
| 94323 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94324 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94325 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94326 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94327 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94328 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94329 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94330 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94331 | <b>syscmd</b>   | The <b>syscmd</b> macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the <b>sysval</b> macro. The behavior is unspecified if <b>syscmd</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                        |
| 94332 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94333 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94334 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94335 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94336 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94337 | <b>sysval</b>   | The defining text of the <b>sysval</b> macro shall be the exit value of the utility last invoked by the <b>syscmd</b> macro (as a string).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 94338 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94339 | <b>traceon</b>  | The <b>traceon</b> macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 94340 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94341 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94342 | <b>traceoff</b> | The <b>traceoff</b> macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 94343 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94344 | <b>translit</b> | The defining text of the <b>translit</b> macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument. The behavior is unspecified if the '-' character appears within the second or third argument anywhere besides the first or last character. The behavior is unspecified if <b>translit</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                      |
| 94345 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94346 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94347 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94348 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94349 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94350 | <b>undefine</b> | The <b>undefine</b> macro shall delete all definitions (including those preserved using the <b>pushdef</b> macro) of the macros named by its arguments. The behavior is unspecified if <b>undefine</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94351 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94352 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94353 | <b>undivert</b> | The <b>undivert</b> macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. The behavior is unspecified if an argument contains any non-numeric characters.                                                                                                                                                                                                                                                                                                                               |
| 94354 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94355 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94356 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 94357 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

94358 **EXIT STATUS**

94359 The following exit values shall be returned:

94360 0 Successful completion.

94361 &gt;0 An error occurred

94362 If the **m4exit** macro is used, the exit value can be specified by the input file.94363 **CONSEQUENCES OF ERRORS**

94364 Default.

94365 **APPLICATION USAGE**94366 The **defn** macro is useful for renaming macros, especially built-ins.

94367 Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some  
 94368 implementations, division or remainder by zero cause a fatal signal, even if the division occurs  
 94369 on the short-circuited branch of " && " or " || ". Any operation that overflows in signed  
 94370 arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount  
 94371 that is not positive and smaller than the precision is undefined, as is shifting a negative number  
 94372 to the right. Historically, not all implementations obeyed C-language precedence rules: '~' and  
 94373 '! ' were lower than '=='; '==' and '! =' were not lower than '<'; and '| ' was not lower  
 94374 than '^ ' ; the liberal use of " ( ) " can force the desired precedence even with these non-  
 94375 compliant implementations. Furthermore, some traditional implementations treated '^ ' as an  
 94376 exponentiation operator, although most implementations now use " \* \* " as an extension for this  
 94377 purpose.

94378 When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the  
 94379 **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or  
 94380 replace the entire stack of definitions with a single definition (as though by **undefine** and  
 94381 **pushdef**). An application desiring particular behavior for the **define** macro in this case can  
 94382 redefine it accordingly.

94383 Applications should use the **mkstemp** macro instead of the obsolescent **maketemp** macro for  
 94384 creating temporary files.

94385 **EXAMPLES**94386 If the file **m4src** contains the lines:

```
94387 The value of 'VER' is "VER".
94388 #ifdef('VER', 'VER' is defined to be VER., VER is not defined.)
94389 #ifelse(VER, 1, 'VER' is 'VER'.)
94390 #ifelse(VER, 2, 'VER' is 'VER.', 'VER' is not 2.)
94391 #end
```

94392 then the command

94393 `m4 m4src`

94394 or the command:

94395 `m4 -U VER m4src`

94396 produces the output:

```
94397 The value of VER is "VER".
94398 VER is not defined.
94399 VER is not 2.
94400 #end
```

94401 The command:

94402 `m4 -D VER m4src`

94403 produces the output:

94404 The value of VER is "".

94405 VER is defined to be .

94406 VER is not 2.

94407 end

94408 The command:

94409 `m4 -D VER=1 m4src`

94410 produces the output:

94411 The value of VER is "1".

94412 VER is defined to be 1.

94413 VER is 1.

94414 VER is not 2.

94415 end

94416 The command:

94417 `m4 -D VER=2 m4src`

94418 produces the output:

94419 The value of VER is "2".

94420 VER is defined to be 2.

94421 VER is 2.

94422 end

#### 94423 RATIONALE

94424 Historic System V-based behavior treated "\${" in a macro definition as two literal characters.  
 94425 However, this sequence is left unspecified so that implementations may offer extensions such as  
 94426 "\$ {11}" meaning the eleventh positional parameter. Macros can still be defined with  
 94427 appropriate uses of nested quoting to result in a literal "\${" in the output after rescanning  
 94428 removes the nested quotes.

94429 In the `translit` built-in, historic System V-based behavior treated '-' as a literal; GNU behavior  
 94430 treats it as a range. This version of the standard allows either behavior.

#### 94431 FUTURE DIRECTIONS

94432 None.

#### 94433 SEE ALSO

94434 [E99](#)

94435 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 94436 CHANGE HISTORY

94437 First released in Issue 2.

#### 94438 Issue 5

94439 The phrase "the defined text for macros written by the `dumpdef` macro" is added to the  
 94440 description of `STDERR`, and the description of `dumpdef` is updated to indicate that output is  
 94441 written to standard error. The description of `eval` is updated to indicate that the list of excluded  
 94442 C operators excludes unary '&' and '.'. In the description of `ifdef`, the phrase "and it is not

- 94443 defined to be zero" is deleted.
- 94444 **Issue 6**
- 94445 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a ' & ' character in the
- 94446 excepted list.
- 94447 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion
- 94448 buffers.
- 94449 The normative text is reworded to avoid use of the term "must" for application requirements.
- 94450 The Open Group Base Resolution bwg2000-006 is applied.
- 94451 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES
- 94452 section.
- 94453 **Issue 7**
- 94454 Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro
- 94455 obsolescent and adding a new **mkstemp** macro.
- 94456 Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space
- 94457 characters that precede or trail any macro arguments.
- 94458 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
- 94459 apply (options can be interspersed with operands).
- 94460 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 94461 SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED
- 94462 DESCRIPTION.
- 94463 SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED
- 94464 DESCRIPTION.
- 94465 SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED
- 94466 DESCRIPTION.
- 94467 SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "{ "
- 94468 produces unspecified behavior.
- 94469 SD5-XCU-ERN-112 is applied, updating the **changequote** macro.
- 94470 SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED
- 94471 DESCRIPTION and APPLICATION USAGE sections.
- 94472 SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED
- 94473 DESCRIPTION and RATIONALE sections.
- 94474 SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.
- 94475 The *m4* utility is moved from the XSI option to the Base.

**mailx**

Utilities

94476 **NAME**

94477 mailx — process messages

94478 **SYNOPSIS**94479 **Send Mode**94480 mailx [-s *subject*] *address*...94481 **Receive Mode**

94482 UP mailx -e

94483 mailx [-HiNn] [-F] [-u *user*]94484 mailx -f [-HiNn] [-F] [*file*]94485 **DESCRIPTION**

94486 The *mailx* utility provides a message sending and receiving facility. It has two major modes,  
 94487 selected by the options used: Send Mode and Receive Mode.

94488 On systems that do not support the User Portability Utilities option, an application using *mailx*  
 94489 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first  
 94490 character of one or more lines is <tilde> ('~'), all characters in the input message shall appear in  
 94491 the delivered message, but additional characters may be inserted in the message before it is  
 94492 retrieved.

94493 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other  
 94494 interactive features, Receive Mode, described below, also shall be enabled.

94495 **Send Mode**

94496 Send Mode can be used by applications or users to send messages from the text in standard  
 94497 input.

94498 UP **Receive Mode**

94499 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this  
 94500 interactive mode.

94501 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to  
 94502 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the  
 94503 message as it is entered.

94504 Incoming mail shall be stored in one or more unspecified locations for each user, collectively  
 94505 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system  
 94506 mailbox shall be the default place to find new mail. As messages are read, they shall be marked  
 94507 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is  
 94508 called the **mbox** and is normally located in the directory referred to by the *HOME* environment  
 94509 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).  
 94510 Messages shall remain in this file until explicitly removed. When the **-f** option is used to read  
 94511 mail messages from secondary files, messages shall be retained in those files unless specifically  
 94512 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred  
 94513 to in this section as simply “mailboxes”, unless more specific identification is required.

94514 **OPTIONS**

94515 The *mailx* utility shall conform to XBD [Section 12.2](#) (on page 215).

94516 The following options shall be supported. (Only the `-s subject` option shall be required on all  
94517 systems. The other options are required only on systems supporting the User Portability Utilities  
94518 option.)

94519 UP **-e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write  
94520 nothing and exit with a successful return code if there is mail to read.

94521 UP **-f** Read messages from the file named by the *file* operand instead of the system  
94522 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox**  
94523 instead of the system mailbox.

94524 UP **-F** Record the message in a file named after the first recipient. The name is the login-  
94525 name portion of the address found first on the **To:** line in the mail header.  
94526 Overrides the **record** variable, if set (see [Internal Variables in mailx](#), on page 2889).

94527 UP **-H** Write a header summary only.

94528 UP **-i** Ignore interrupts. (See also **ignore**.)

94529 UP **-n** Do not initialize from the system default start-up file. See the EXTENDED  
94530 DESCRIPTION section.

94531 UP **-N** Do not write an initial header summary.

94532 **-s subject** Set the **Subject** header field to *subject*. All characters in the *subject* string shall  
94533 appear in the delivered message. The results are unspecified if *subject* is longer  
94534 than {LINE\_MAX} – 10 bytes or contains a <newline>.

94535 UP **-u user** Read the system mailbox of the login name *user*. This shall only be successful if  
94536 the invoking user has appropriate privileges to read the system mailbox of that  
94537 user.

94538 **OPERANDS**

94539 The following operands shall be supported:

94540 *address* Addressee of message. When **-n** is specified and no user start-up files are accessed  
94541 (see the EXTENDED DESCRIPTION section), the user or application shall ensure  
94542 this is an address to pass to the mail delivery system. Any system or user start-up  
94543 files may enable aliases (see **alias** under [Commands in mailx](#), on page 2892) that  
94544 may modify the form of *address* before it is passed to the mail delivery system.

94545 UP *file* A pathname of a file to be read instead of the system mailbox when **-f** is specified.  
94546 The meaning of the *file* option-argument shall be affected by the contents of the  
94547 **folder** internal variable; see [Internal Variables in mailx](#) (on page 2889).

94548 **STDIN**

94549 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the  
94550 UP message to be delivered to the specified addresses. When in Receive Mode, user commands  
94551 shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard  
94552 input lines beginning with a <tilde> ('~') character produce unspecified results.

94553 UP If the User Portability Utilities option is supported, then in both Send and Receive Modes,  
94554 standard input lines beginning with the escape character (usually <tilde> ('~')) shall affect  
94555 processing as described in [Command Escapes in mailx](#) (on page 2901).

## 94556 INPUT FILES

94557 When *mailx* is used as described by this volume of POSIX.1-2008, the *file* option-argument (see  
 94558 the `-f` option) and the **mbox** shall be text files containing mail messages, formatted as described  
 94559 in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a  
 94560 file.

## 94561 ENVIRONMENT VARIABLES

94562 UP Some of the functionality described in this section shall be provided on implementations that  
 94563 support the User Portability Utilities option as described in the text, and is not further shaded  
 94564 for this option.

94565 The following environment variables shall affect the execution of *mailx*:

94566 *DEAD* Determine the pathname of the file in which to save partial messages in case of  
 94567 interrupts or delivery errors. The default shall be **dead.letter** in the directory  
 94568 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is  
 94569 unspecified if the User Portability Utilities option is not supported and *DEAD* is  
 94570 not defined with the value **/dev/null**.

94571 *EDITOR* Determine the name of a utility to invoke when the **edit** (see [Commands in mailx](#),  
 94572 on page 2892) or **~e** (see [Command Escapes in mailx](#), on page 2901) command is  
 94573 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The  
 94574 effects of this variable are unspecified if the User Portability Utilities option is not  
 94575 supported.

94576 *HOME* Determine the pathname of the user's home directory.

94577 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 94578 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization  
 94579 variables used to determine the values of locale categories.)

94580 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 94581 internationalization variables.

94582 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 94583 characters (for example, single-byte as opposed to multi-byte characters in  
 94584 arguments and input files) and the handling of case-insensitive address and  
 94585 header-field comparisons.

94586 *LC\_TIME* This variable may determine the format and contents of the date and time strings  
 94587 written by *mailx*. This volume of POSIX.1-2008 specifies the effects of this variable  
 94588 only for systems supporting the User Portability Utilities option.

94589 *LC\_MESSAGES*

94590 Determine the locale that should be used to affect the format and contents of  
 94591 diagnostic messages written to standard error and informative messages written to  
 94592 standard output.

94593 *LISTER* Determine a string representing the command for writing the contents of the  
 94594 **folder** directory to standard output when the **folders** command is given (see  
 94595 **folders** in [Commands in mailx](#), on page 2892). Any string acceptable as a  
 94596 *command\_string* operand to the *sh -c* command shall be valid. If this variable is null  
 94597 or not set, the output command shall be *ls*. The effects of this variable are  
 94598 unspecified if the User Portability Utilities option is not supported.

94599 *MAILRC* Determine the pathname of the start-up file. The default shall be **.mailrc** in the  
 94600 directory referred to by the *HOME* environment variable. The behavior of *mailx* is  
 94601 unspecified if the User Portability Utilities option is not supported and *MAILRC* is

|       |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 94602 |                            | not defined with the value <code>/dev/null</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94603 | <i>MBOX</i>                | Determine a pathname of the file to save messages from the system mailbox that have been read. The <code>exit</code> command shall override this function, as shall saving the message explicitly in another file. The default shall be <code>mbox</code> in the directory named by the <i>HOME</i> variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                        |
| 94604 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94605 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94606 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94607 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94608 | XSI                        | <i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 94609 | <i>PAGER</i>               | Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a <i>command_string</i> operand to the <code>sh -c</code> command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the <i>mailx</i> internal variable <code>cr</code> is set to a value less the number of lines in the message; see <a href="#">Internal Variables in mailx</a> (on page 2889). If the <i>PAGER</i> variable is null or not set, the paginator shall be either <code>more</code> or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported. |
| 94610 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94611 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94612 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94613 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94614 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94615 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94616 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94617 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94618 | <i>SHELL</i>               | Determine the name of a preferred command interpreter. The default shall be <code>sh</code> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 94619 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94620 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94621 | <i>TERM</i>                | If the internal variable <code>screen</code> is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of headers. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                              |
| 94622 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94623 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94624 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94625 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94626 | <i>TZ</i>                  | This variable may determine the timezone used to calculate date and time strings written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 94627 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94628 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94629 | <i>VISUAL</i>              | Determine a pathname of a utility to invoke when the <code>visual</code> command (see <a href="#">Commands in mailx</a> , on page 2892) or <code>~v</code> command-escape (see <a href="#">Command Escapes in mailx</a> , on page 2901) is used. If this variable is null or not set, the full-screen editor shall be <code>vi</code> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                              |
| 94630 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94631 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94632 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94633 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94634 | <b>ASYNCHRONOUS EVENTS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94635 |                            | When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 94636 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94637 | UP                         | In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is received:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94638 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94639 | UP                         | 1. If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 94640 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94641 |                            | 2. If in input mode:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 94642 | UP                         | a. If <code>ignore</code> is set, <i>mailx</i> shall write " <code>@\n</code> ", discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 94643 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 94644 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

- 94645 UP            b. If the interrupt was received while sending mail, either when in Receive Mode or  
 94646            in Send Mode, a message shall be written, and another subsequent interrupt, with  
 94647            no other intervening characters typed, shall be required to abort the mail message.  
 94648 UP            If in Receive Mode and another interrupt is received, a command-mode prompt  
 94649            shall be written. If in Send Mode and another interrupt is received, *mailx* shall  
 94650            terminate with a non-zero status.

94651            In both cases listed in item b, if the message is not empty:

- 94652 UP            i. If **save** is enabled and the file named by *DEAD* can be created, the message  
 94653            shall be written to the file named by *DEAD*. If the file exists, the message  
 94654            shall be written to replace the contents of the file.
- 94655 UP            ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the  
 94656            message shall not be saved.

94657            The *mailx* utility shall take the standard action for all other signals.

#### 94658 **STDOUT**

94659            In command and input modes, all output, including prompts and messages, shall be written to  
 94660            standard output.

#### 94661 **STDERR**

94662            The standard error shall be used only for diagnostic messages.

#### 94663 **OUTPUT FILES**

94664            Various *mailx* commands and command escapes can create or add to files, including the **mbox**,  
 94665            the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of  
 94666            POSIX.1-2008, these files shall be text files, formatted as follows:

94667            line beginning with **From<space>**  
 94668            [one or more *header-lines*; see [Commands in mailx](#) (on page 2892)]  
 94669            *empty line*  
 94670            [zero or more *body lines*  
 94671            *empty line*]  
 94672            [line beginning with **From<space>...]**

94673            where each message begins with the **From <space>** line shown, preceded by the beginning of  
 94674            the file or an empty line. (The **From <space>** line is considered to be part of the message header,  
 94675            but not one of the header-lines referred to in [Commands in mailx](#) (on page 2892); thus, it shall  
 94676            not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the  
 94677            **From <space>** line and any additional header lines are unspecified, except that none shall be  
 94678            empty. The format of a message body line is also unspecified, except that no line following an  
 94679            empty line shall start with **From <space>**; *mailx* shall modify any such user-entered message  
 94680            body lines (following an empty line and beginning with **From <space>**) by adding one or more  
 94681            characters to precede the 'F'; it may add these characters to **From <space>** lines that are not  
 94682            preceded by an empty line.

94683            When a message from the system mailbox or entered by the user is not a text file, it is  
 94684            implementation-defined how such a message is stored in files written by *mailx*.

#### 94685 **EXTENDED DESCRIPTION**

94686 UP            The functionality in the entire EXTENDED DESCRIPTION section shall be provided on  
 94687            implementations supporting the User Portability Utilities option. The functionality described in  
 94688            this section shall be provided on implementations that support the User Portability Utilities  
 94689            option (and the rest of this section is not further shaded for this option).

94690 The *mailx* utility need not support for all character encodings in all circumstances. For example,  
 94691 inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not  
 94692 be portable to non-internationalized systems, and so on. Under these circumstances, it is  
 94693 recommended that only characters defined in the ISO/IEC 646:1991 standard International  
 94694 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

94695 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of  
 94696 header-summary lines (if `-N` was not specified and there are messages, see below), followed by  
 94697 a prompt indicating that *mailx* can accept regular commands (see [Commands in mailx](#), on page  
 94698 2892); this is termed *command mode*. The page of header-summary lines shall contain the first  
 94699 new message if there are new messages, or the first unread message if there are unread  
 94700 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and  
 94701 standard input is a terminal, if no subject is specified on the command line and the `asksub`  
 94702 variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input  
 94703 mode. This input mode shall also be entered when using one of the Receive Mode synopsis  
 94704 forms and a reply or new message is composed using the `reply`, `Reply`, `followup`, `Followup`, or  
 94705 `mail` commands and standard input is a terminal. When the message is typed and the end of the  
 94706 message is encountered, the message shall be passed to the mail delivery software. Commands  
 94707 can be entered by beginning a line with the escape character (by default, `<tilde>` (`' ~ '`)) followed  
 94708 by a single command letter and optional arguments. See [Commands in mailx](#) (on page 2892) for  
 94709 a summary of these commands. It is unspecified what effect these commands will have if  
 94710 standard input is not a terminal when a message is entered using either the Send Mode  
 94711 synopsis, or the Read Mode commands `reply`, `Reply`, `followup`, `Followup`, or `mail`.

94712 **Note:** For notational convenience, this section uses the default escape character, `<tilde>`, in all  
 94713 references and examples.

94714 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal  
 94715 variables. These are flags and valued parameters that can be set and cleared via the *mailx* `set`  
 94716 and `unset` commands.

94717 Regular commands are of the form:

94718 `[command] [msglist] [argument ...]`

94719 If no *command* is specified in command mode, `next` shall be assumed. In input mode, commands  
 94720 shall be recognized by the escape character, and lines not treated as commands shall be taken as  
 94721 input for the message.

94722 In command mode, each message shall be assigned a sequential number, starting with 1.

94723 All messages have a state that shall affect how they are displayed in the header summary and  
 94724 how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a  
 94725 *current* message, which shall be marked by a `'>'` at the beginning of a line in the header  
 94726 summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current  
 94727 message shall be the first new message, if there is a new message, or the first unread message if  
 94728 there is an unread message, or the first message if there are any messages, or unspecified if there  
 94729 are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*)  
 94730 or an optional single message (*message*) on which to operate shall leave the current message set  
 94731 to the highest-numbered message of the messages specified, unless the command deletes  
 94732 messages, in which case the current message shall be set to the first undeleted message (that is, a  
 94733 message not in the deleted state) after the highest-numbered message deleted by the command,  
 94734 if one exists, or the first undeleted message before the highest-numbered message deleted by the  
 94735 command, if one exists, or to an unspecified value if there are no remaining undeleted messages.  
 94736 All messages shall be in one of the following states:

## mailx

## Utilities

|       |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 94737 | <i>new</i>       | The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state <i>new</i> when <i>mailx</i> quits shall be retained in the system mailbox.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 94738 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94739 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94740 | <i>unread</i>    | The message has been present in the system mailbox for more than one invocation of <i>mailx</i> and has not been viewed by the user or moved to any other state. Messages in state <i>unread</i> when <i>mailx</i> quits shall be retained in the system mailbox.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94741 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94742 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94743 | <i>read</i>      | The message has been processed by one of the following commands: <b>~f</b> , <b>~m</b> , <b>~F</b> , <b>M</b> , <b>copy</b> , <b>mbox</b> , <b>next</b> , <b>pipe</b> , <b>print</b> , <b>Print</b> , <b>top</b> , <b>type</b> , <b>Type</b> , <b>undelete</b> . The <b>delete</b> , <b>dp</b> , and <b>dt</b> commands may also cause the next message to be marked as <i>read</i> , depending on the value of the <b>autoprint</b> variable. Messages that are in the system mailbox and in state <i>read</i> when <i>mailx</i> quits shall be saved in the <b>mbox</b> , unless the internal variable <b>hold</b> was set. Messages that are in the <b>mbox</b> or in a secondary mailbox and in state <i>read</i> when <i>mailx</i> quits shall be retained in their current location. |
| 94744 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94745 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94746 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94747 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94748 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94749 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94750 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94751 | <i>deleted</i>   | The message has been processed by one of the following commands: <b>delete</b> , <b>dp</b> , <b>dt</b> . Messages in state <i>deleted</i> when <i>mailx</i> quits shall be deleted. Deleted messages shall be ignored until <i>mailx</i> quits or changes mailboxes or they are specified to the <b>undelete</b> command; for example, the message specification <i>/string</i> shall only search the subject lines of messages that have not yet been deleted, unless the command operating on the list of messages is <b>undelete</b> . No deleted message or deleted message header shall be displayed by any <i>mailx</i> command other than <b>undelete</b> .                                                                                                                         |
| 94752 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94753 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94754 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94755 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94756 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94757 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94758 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94759 | <i>preserved</i> | The message has been processed by a <b>preserve</b> command. When <i>mailx</i> quits, the message shall be retained in its current location.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94760 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94761 | <i>saved</i>     | The message has been processed by one of the following commands: <b>save</b> or <b>write</b> . If the current mailbox is the system mailbox, and the internal variable <b>keepsave</b> is set, messages in the state <i>saved</i> shall be saved to the file designated by the <b>MBOX</b> variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is the system mailbox, messages in the state <i>saved</i> shall be deleted from the current mailbox, when the <b>quit</b> or <b>file</b> command is used to exit the current mailbox.                                                                                                                                                                                                                                  |
| 94762 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94763 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94764 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94765 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94766 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94767 |                  | The header-summary line for each message shall indicate the state of the message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94768 |                  | Many commands take an optional list of messages ( <i>msglist</i> ) on which to operate, which defaults to the current message. A <i>msglist</i> is a list of message specifications separated by <blank> characters, which can include:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 94769 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94770 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94771 | <i>n</i>         | Message number <i>n</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 94772 | <b>+</b>         | The next undeleted message, or the next deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 94773 |                  | The next previous undeleted message, or the next previous deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 94774 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 94775 | <b>.</b>         | The current message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 94776 | <b>^</b>         | The first undeleted message, or the first deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 94777 | <b>\$</b>        | The last message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94778 | <b>*</b>         | All messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

|       |                |                                                                                                              |
|-------|----------------|--------------------------------------------------------------------------------------------------------------|
| 94779 | <i>n-m</i>     | An inclusive range of message numbers.                                                                       |
| 94780 | <i>address</i> | All messages from <i>address</i> ; any address as shown in a header summary shall be matchable in this form. |
| 94781 |                |                                                                                                              |
| 94782 | <i>/string</i> | All messages with <i>string</i> in the subject line (case ignored).                                          |
| 94783 | <i>:c</i>      | All messages of type <i>c</i> , where <i>c</i> shall be one of:                                              |
| 94784 | <i>d</i>       | Deleted messages.                                                                                            |
| 94785 | <i>n</i>       | New messages.                                                                                                |
| 94786 | <i>o</i>       | Old messages (any not in state <i>read</i> or <i>new</i> ).                                                  |
| 94787 | <i>r</i>       | Read messages.                                                                                               |
| 94788 | <i>u</i>       | Unread messages.                                                                                             |

94789 Other commands take an optional message (*message*) on which to operate, which defaults to the  
 94790 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more  
 94791 than one message is specified, only the first shall be operated on.

94792 Other arguments are usually arbitrary strings whose usage depends on the command involved.

### 94793 **Start-Up in mailx**

94794 At start-up time, *mailx* shall take the following steps in sequence:

- 94795 1. Establish all variables at their stated default values.
- 94796 2. Process command line options, overriding corresponding default values.
- 94797 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables  
 94798 that are present in the environment, overriding the corresponding default values.
- 94799 4. Read *mailx* commands from an unspecified system start-up file, unless the *-n* option is  
 94800 given, to initialize any internal *mailx* variables and aliases.
- 94801 5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

94802 Most regular *mailx* commands are valid inside start-up files, the most common use being to set  
 94803 up initial display options and alias lists. The following commands shall be invalid in the start-up  
 94804 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.  
 94805 Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and  
 94806 a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of  
 94807 the lines in the start-up file.

94808 A blank line in a start-up file shall be ignored.

### 94809 **Internal Variables in mailx**

94810 The following variables are internal *mailx* variables. Each internal variable can be set via the  
 94811 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase  
 94812 variables.

94813 In the following list, variables shown as:

94814 *variable*

94815 represent Boolean values. Variables shown as:

94816 *variable=value*

## mailx

## Utilities

- 94817 shall be assigned string or numeric values. For string values, the rules in [Commands in mailx](#)  
 94818 (on page 2892) concerning filenames and quoting shall also apply.
- 94819 The defaults specified here may be changed by the unspecified system start-up file unless the  
 94820 user specifies the `-n` option.
- 94821 **allnet** All network names whose login name components match shall be treated as  
 94822 identical. This shall cause the `msglist` message specifications to behave similarly.  
 94823 The default shall be **noallnet**. See also the **alternates** command and the **metoo**  
 94824 variable.
- 94825 **append** Append messages to the end of the **mbox** file upon termination instead of placing  
 94826 them at the beginning. The default shall be **noappend**. This variable shall not  
 94827 affect the **save** command when saving to **mbox**.
- 94828 **ask, asksub** Prompt for a subject line on outgoing mail if one is not specified on the command  
 94829 line with the `-s` option. The **ask** and **asksub** forms are synonyms; the system shall  
 94830 refer to **asksub** and **noasksub** in its messages, but shall accept **ask** and **noask** as  
 94831 user input to mean **asksub** and **noasksub**. It shall not be possible to set both **ask**  
 94832 and **noasksub**, or **noask** and **asksub**. The default shall be **asksub**, but no  
 94833 prompting shall be done if standard input is not a terminal.
- 94834 **askbcc** Prompt for the blind copy list. The default shall be **noaskbcc**.
- 94835 **askcc** Prompt for the copy list. The default shall be **noaskcc**.
- 94836 **autoprint** Enable automatic writing of messages after **delete** and **undelete** commands. The  
 94837 default shall be **noautoprint**.
- 94838 **bang** Enable the special-case treatment of <exclamation-mark> characters ('!') in  
 94839 escape command lines; see the **escape** command and [Command Escapes in mailx](#)  
 94840 (on page 2901). The default shall be **nobang**, disabling the expansion of '!' in the  
 94841 `command` argument to the `!` command and the `~!command` escape.
- 94842 **cmd=command**  
 94843 Set the default command to be invoked by the **pipe** command. The default shall be  
 94844 **nocmd**.
- 94845 **crt=number** Pipe messages having more than `number` lines through the command specified by  
 94846 the value of the `PAGER` variable. The default shall be **nocrt**. If it is set to null, the  
 94847 value used is implementation-defined.
- 94848 XSI **debug** Enable verbose diagnostics for debugging. Messages are not delivered. The  
 94849 default shall be **nodebug**.
- 94850 **dot** When **dot** is set, a <period> on a line by itself during message input from a  
 94851 terminal shall also signify end-of-file (in addition to normal end-of-file). The  
 94852 default shall be **nodot**. If **ignoreeof** is set (see below), a setting of **nodot** shall be  
 94853 ignored and the <period> is the only method to terminate input mode.
- 94854 **escape=c** Set the command escape character to be the character 'c'. By default, the  
 94855 command escape character shall be <tilde>. If **escape** is unset, <tilde> shall be  
 94856 used; if it is set to null, command escaping shall be disabled.
- 94857 **flipr** Reverse the meanings of the **R** and **r** commands. The default shall be **noflipr**.
- 94858 **folder=directory**  
 94859 The default directory for saving mail files. User-specified filenames beginning with  
 94860 a <plus-sign> ('+') shall be expanded by preceding the filename with this

|       |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 94861 |                            | directory name to obtain the real pathname. If <i>directory</i> does not start with a <slash> (' / '), the contents of <i>HOME</i> shall be prefixed to it. The default shall be <b>nofolder</b> . If <b>folder</b> is unset or set to null, user-specified filenames beginning with '+' shall refer to files in the current directory that begin with the literal '+' character. See also <b>outfolder</b> below. The <b>folder</b> value need not affect the processing of the files named in <i>MBOX</i> and <i>DEAD</i> . |
| 94862 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94863 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94864 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94865 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94866 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94867 | <b>header</b>              | Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The default shall be <b>header</b> .                                                                                                                                                                                                                                                                                                                                                                                                         |
| 94868 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94869 | <b>hold</b>                | Preserve all messages that are read in the system mailbox instead of putting them in the <b>mbox</b> save file. The default shall be <b>nohold</b> .                                                                                                                                                                                                                                                                                                                                                                          |
| 94870 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94871 | <b>ignore</b>              | Ignore interrupts while entering messages. The default shall be <b>noignore</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 94872 | <b>ignoreeof</b>           | Ignore normal end-of-file during message input. Input can be terminated only by entering a <period> (' . ') on a line by itself or by the ~. command escape. The default shall be <b>noignoreeof</b> . See also <b>dot</b> above.                                                                                                                                                                                                                                                                                             |
| 94873 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94874 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94875 | <b>indentprefix=string</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94876 |                            | A string that shall be added as a prefix to each line that is inserted into the message by the ~m command escape. This variable shall default to one <tab>.                                                                                                                                                                                                                                                                                                                                                                   |
| 94877 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94878 | <b>keep</b>                | When a system mailbox, secondary mailbox, or <b>mbox</b> is empty, truncate it to zero length instead of removing it. The default shall be <b>nokeep</b> .                                                                                                                                                                                                                                                                                                                                                                    |
| 94879 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94880 | <b>keepsave</b>            | Keep the messages that have been saved from the system mailbox into other files in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default shall be <b>nokeepsave</b> .                                                                                                                                                                                                                                                                                                                       |
| 94881 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94882 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94883 | <b>metoo</b>               | Suppress the deletion of the login name of the user from the recipient list when replying to a message or sending to a group. The default shall be <b>nometoo</b> .                                                                                                                                                                                                                                                                                                                                                           |
| 94884 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94885 | XSI <b>onehop</b>          | When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). The default shall be <b>noonehop</b> .                                                                                                  |
| 94886 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94887 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94888 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94889 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94890 | <b>outfolder</b>           | Cause the files used to record outgoing messages to be located in the directory specified by the <b>folder</b> variable unless the pathname is absolute. The default shall be <b>nooutfolder</b> . See the <b>record</b> variable.                                                                                                                                                                                                                                                                                            |
| 94891 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94892 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94893 | <b>page</b>                | Insert a <form-feed> after each message sent through the pipe created by the <b>pipe</b> command. The default shall be <b>nopage</b> .                                                                                                                                                                                                                                                                                                                                                                                        |
| 94894 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94895 | <b>prompt=string</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94896 |                            | Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if <b>noprompt</b> is set, no prompting shall occur. The default shall be to prompt with the string "? ".                                                                                                                                                                                                                                                                                                                                          |
| 94897 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94898 | <b>quiet</b>               | Refrain from writing the opening message and version when entering <i>mailx</i> . The default shall be <b>noquiet</b> .                                                                                                                                                                                                                                                                                                                                                                                                       |
| 94899 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94900 | <b>record=file</b>         | Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be <b>norecord</b> . See also <b>outfolder</b> above.                                                                                                                                                                                                                                                                                                                                                                                  |
| 94901 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 94902 | <b>save</b>                | Enable saving of messages in the dead-letter file on interrupt or delivery error. See the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be <b>save</b> .                                                                                                                                                                                                                                                                                                                                   |
| 94903 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

- 94904 **screen=number**
- 94905 Set the number of lines in a screenful of headers for the **headers** and **z** commands.
- 94906 If **screen** is not specified, a value based on the terminal type identified by the
- 94907 *TERM* environment variable, the window size, the baud rate, or some combination
- 94908 of these shall be used.
- 94909 **sendwait** Wait for the background mailer to finish before returning. The default shall be
- 94910 **nosendwait**.
- 94911 **showto** When the sender of the message was the user who is invoking *mailx*, write the
- 94912 information from the **To:** line instead of the **From:** line in the header summary. The
- 94913 default shall be **noshowto**.
- 94914 **sign=string** Set the variable inserted into the text of a message when the **~a** command escape is
- 94915 given. The default shall be **nosign**. The character sequences **'\t'** and **'\n'** shall
- 94916 be recognized in the variable as **<tab>** and **<newline>** characters, respectively. (See
- 94917 also **~i** in **Command Escapes in mailx** (on page 2901).)
- 94918 **Sign=string** Set the variable inserted into the text of a message when the **~A** command escape is
- 94919 given. The default shall be **noSign**. The character sequences **'\t'** and **'\n'** shall
- 94920 be recognized in the variable as **<tab>** and **<newline>** characters, respectively.
- 94921 **toplines=number**
- 94922 Set the number of lines of the message to write with the **top** command. The default
- 94923 shall be 5.
- 94924 **Commands in mailx**
- 94925 The following *mailx* commands shall be provided. In the following list, header refers to lines
- 94926 from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines
- 94927 within the header that begin with one or more non-white-space characters, immediately
- 94928 followed by a **<colon>** and white space and continuing until the next line beginning with a non-
- 94929 white-space character or an empty line. Header-field refers to the portion of a header line prior
- 94930 to the first **<colon>** in that line.
- 94931 For each of the commands listed below, the command can be entered as the abbreviation (those
- 94932 characters in the Synopsis command word preceding the **' [ '**), the full command (all characters
- 94933 shown for the command word, omitting the **' [ '** and **' ] '**), or any truncation of the full
- 94934 command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the
- 94935 Synopsis) can be entered as **ex**, **exi**, or **exit**.
- 94936 The arguments to commands can be quoted, using the following methods:
- 94937 • An argument can be enclosed between paired double-quotes (**" "**) or single-quotes (**' '**);
  - 94938 any white space, shell word expansion, or **<backslash>** characters within the quotes shall
  - 94939 be treated literally as part of the argument. A double-quote shall be treated literally within
  - 94940 single-quotes and *vice versa*. These special properties of the **<quotation-mark>** characters
  - 94941 shall occur only when they are paired at the beginning and end of the argument.
  - 94942 • A **<backslash>** outside of the enclosing quotes shall be discarded and the following
  - 94943 character treated literally as part of the argument.
  - 94944 • An unquoted **<backslash>** at the end of a command line shall be discarded and the next
  - 94945 line shall continue the command.

94946 Filenames, where expected, shall be subjected to the following transformations, in sequence:

- 94947 • If the filename begins with an unquoted <plus-sign>, and the **folder** variable is defined
- 94948 (see the **folder** variable), the <plus-sign> shall be replaced by the value of the **folder**
- 94949 variable followed by a <slash>. If the **folder** variable is unset or is set to null, the filename
- 94950 shall be unchanged.
- 94951 • Shell word expansions shall be applied to the filename (see Section 2.6, on page 2305). If
- 94952 more than a single pathname results from this expansion and the command is expecting
- 94953 one file, the effects are unspecified.

#### 94954 **Declare Aliases**

94955 *Synopsis:*    a[lias] [alias [address...]]  
 94956               g[roup] [alias [address...]]

94957 Add the given addresses to the alias specified by *alias*. The names shall be substituted when  
 94958 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other  
 94959 recipients addressed indirectly through the **reply** command shall not be substituted in this  
 94960 manner). Mail address alias substitution shall apply only when the alias string is used as a full  
 94961 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If  
 94962 no arguments are given, write a listing of the current aliases to standard output. If only an *alias*  
 94963 argument is given, write a listing of the specified alias to standard output. These listings need  
 94964 not reflect the same order of addresses that were entered.

#### 94965 **Declare Alternatives**

94966 *Synopsis:*    alt[ernates] name...

94967 (See also the **metoo** variable.) Declare a list of alternative names for the user's login. When  
 94968 responding to a message, these names shall be removed from the list of recipients for the  
 94969 response. The comparison of names shall be in a case-insensitive manner. With no arguments,  
 94970 **alternates** shall write the current list of alternative names.

#### 94971 **Change Current Directory**

94972 *Synopsis:*    cd [directory]  
 94973               ch[dir] [directory]

94974 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

#### 94975 **Copy Messages**

94976 *Synopsis:*    c[opy] [file]  
 94977               c[opy] [msglist] file  
 94978               C[opy] [msglist]

94979 Copy messages to the file named by the pathname *file* without marking the messages as saved.  
 94980 Otherwise, it shall be equivalent to the **save** command.

94981 In the capitalized form, save the specified messages in a file whose name is derived from the  
 94982 author of the message to be saved, without marking the messages as saved. Otherwise, it shall  
 94983 be equivalent to the **Save** command.

94984 **Delete Messages**94985 *Synopsis:* d[etele] [*msglist*]

94986 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see  
 94987 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there  
 94988 are messages remaining after the **delete** command, the current message shall be written as  
 94989 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be  
 94990 written.

94991 **Discard Header Fields**94992 *Synopsis:* di[scard] [*header-field...*]94993 ig[nore] [*header-field...*]

94994 Suppress the specified header fields when writing messages. Specified *header-fields* shall be  
 94995 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and  
 94996 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall  
 94997 override this command. The comparison of header fields shall be in a case-insensitive manner. If  
 94998 no arguments are specified, write a list of the currently suppressed header fields to standard  
 94999 output; the listing need not reflect the same order of header fields that were entered.

95000 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.95001 **Delete Messages and Display**95002 *Synopsis:* dp [*msglist*]95003 dt [*msglist*]

95004 Delete the specified messages as described for the **delete** command, except that the **autoprint**  
 95005 variable shall have no effect, and the current message shall be written only if it was set to a  
 95006 message after the last message deleted by the command. Otherwise, an informational message  
 95007 to the effect that there are no further messages in the mailbox shall be written, followed by the  
 95008 *mailx* prompt.

95009 **Echo a String**95010 *Synopsis:* ec[ho] *string ...*95011 Echo the given strings, equivalent to the shell *echo* utility.95012 **Edit Messages**95013 *Synopsis:* e[dit] [*msglist*]

95014 Edit the given messages. The messages shall be placed in a temporary file and the utility named  
 95015 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is  
 95016 unspecified.

95017 The **edit** command does not modify the contents of those messages in the mailbox.

95018 **Exit**95019 *Synopsis:*    ex[it]

95020            x[it]

95021 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also  
95022 **quit**).95023 **Change Folder**95024 *Synopsis:*    fi[le] [file]

95025            fold[er] [file]

95026 Quit (see the **quit** command) from the current file of messages and read in the file named by the  
95027 pathname *file*. If no argument is given, the name and status of the current mailbox shall be  
95028 written.95029 Several unquoted special characters shall be recognized when used as *file* names, with the  
95030 following substitutions:

95031 %            The system mailbox for the invoking user.

95032 %*user*       The system mailbox for *user*.

95033 #            The previous file.

95034 &            The current **mbox**.95035 +*file*       The named file in the **folder** directory. (See the **folder** variable.)

95036 The default file shall be the current mailbox.

95037 **Display List of Folders**95038 *Synopsis:*    folders95039 Write the names of the files in the directory set by the **folder** variable. The command specified  
95040 by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES  
95041 section).95042 **Follow Up Specified Messages**95043 *Synopsis:*    fo[llowup] [message]

95044            F[ollowup] [msglist]

95045 In the lowercase form, respond to a message, recording the response in a file whose name is  
95046 derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.95047 In the capitalized form, respond to the first message in the *msglist*, sending the message to the  
95048 author of each message in the *msglist*. The subject line shall be taken from the first message and  
95049 the response shall be recorded in a file whose name is derived from the author of the first  
95050 message. See also the **Save** and **Copy** commands and **outfolder**.95051 Both forms shall override the **record** variable, if set.

95052 **Display Header Summary for Specified Messages**95053 *Synopsis:* f[rom] [msglist]

95054 Write the header summary for the specified messages.

95055 **Display Header Summary**95056 *Synopsis:* h[eaders] [message]

95057 Write the page of headers that includes the message specified. If the *message* argument is not  
 95058 specified, the current message shall not change. However, if the *message* argument is specified,  
 95059 the current message shall become the message that appears at the top of the page of headers that  
 95060 includes the message specified. The **screen** variable sets the number of headers per page. See  
 95061 also the **z** command.

95062 **Help**95063 *Synopsis:* hel[p]

95064 ?

95065 Write a summary of commands.

95066 **Hold Messages**95067 *Synopsis:* ho[ld] [msglist]

95068 pre[serve] [msglist]

95069 Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall  
 95070 override any commands that might previously have marked the messages to be deleted. During  
 95071 the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*  
 95072 marking of a message.

95073 **Execute Commands Conditionally**95074 *Synopsis:* i[f] s|r95075 *mail-commands*

95076 el[se]

95077 *mail-commands*

95078 en[dif]

95079 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an  
 95080 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be  
 95081 executed only in Receive Mode.

95082 **List Available Commands**95083 *Synopsis:* l[ist]

95084 Write a list of all commands available. No explanation shall be given.

95085 **Mail a Message**95086 *Synopsis:* m[ail] address...

95087 Mail a message to the specified addresses or aliases.

95088 **Direct Messages to mbox**95089 *Synopsis:* mb[ox] [msglist]95090 Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.  
95091 See **MBOX**. See also the **exit** and **quit** commands.95092 **Process Next Specified Message**95093 *Synopsis:* n[ext] [message]95094 If the current message has not been written (for example, by the **print** command) since *mailx*  
95095 started or since any other message was the current message, behave as if the **print** command  
95096 was entered. Otherwise, if there is an undeleted message after the current message, make it the  
95097 current message and behave as if the **print** command was entered. Otherwise, an informational  
95098 message to the effect that there are no further messages in the mailbox shall be written, followed  
95099 by the *mailx* prompt. Should the current message location be the result of an immediately  
95100 preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** will act as if the current message has  
95101 already been written.95102 **Pipe Message**95103 *Synopsis:* pi[pe] [[msglist] command]  
95104 | [[msglist] command]95105 Pipe the messages through the given *command* by invoking the command interpreter specified  
95106 by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure  
95107 that the command is given as a single argument. Quoting, described previously, can be used to  
95108 accomplish this. If no arguments are given, the current message shall be piped through the  
95109 command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>  
95110 shall be inserted after each message.95111 **Display Message with Headers**95112 *Synopsis:* P[rint] [msglist]  
95113 T[ype] [msglist]95114 Write the specified messages, including all header lines, to standard output. Override  
95115 suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages  
95116 longer than the number of lines specified by the **crt** variable shall be paged through the  
95117 command specified by the *PAGER* environment variable.95118 **Display Message**95119 *Synopsis:* p[rint] [msglist]  
95120 t[ype] [msglist]95121 Write the specified messages to standard output. If **crt** is set, the messages longer than the  
95122 number of lines specified by the **crt** variable shall be paged through the command specified by  
95123 the *PAGER* environment variable.

95124 **Quit**

95125 *Synopsis:*    q[uit]  
 95126            end-of-file

95127 Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system  
 95128 mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless  
 95129 **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages  
 95130 in the mailbox.

95131 **Reply to a Message List**

95132 *Synopsis:*    R[eply] [msglist]  
 95133            R[espond] [msglist]

95134 Mail a reply message to the sender of each message in the *msglist*. The subject line shall be  
 95135 formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject  
 95136 from the first message. If **record** is set to a filename, the response shall be saved at the end of that  
 95137 file.

95138 See also the **flipr** variable.

95139 **Reply to a Message**

95140 *Synopsis:*    r[eply] [message]  
 95141            r[espond] [message]

95142 Mail a reply message to all recipients included in the header of the message. The subject line  
 95143 shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the  
 95144 subject from the message. If **record** is set to a filename, the response shall be saved at the end of  
 95145 that file.

95146 See also the **flipr** variable.

95147 **Retain Header Fields**

95148 *Synopsis:*    ret[ain] [header-field...]

95149 Retain the specified header fields when writing messages. This command shall override all  
 95150 **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive  
 95151 manner. If no arguments are specified, write a list of the currently retained header fields to  
 95152 standard output; the listing need not reflect the same order of header fields that were entered.

95153 **Save Messages**

95154 *Synopsis:*    s[ave] [file]  
 95155            s[ave] [msglist] file  
 95156            S[ave] [msglist]

95157 Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file*  
 95158 argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be  
 95159 appended to the file. The message shall be put in the state *saved*, and shall behave as specified in  
 95160 the description of the *saved* state when the current mailbox is exited by the **quit** or **file**  
 95161 command.

95162 In the capitalized form, save the specified messages in a file whose name is derived from the  
 95163 author of the first message. The name of the file shall be taken to be the author's name with all  
 95164 network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and

95165 **outfolder** variable.

### 95166 **Set Variables**

95167 *Synopsis:* `se[t] [name=[string]] ... [name=number ...] [noname ...]`

95168 Define one or more variables called *name*. The variable can be given a null, string, or numeric  
 95169 value. Quoting and <backslash>-escapes can occur anywhere in *string*, as described previously,  
 95170 as if the *string* portion of the argument were the entire argument. The forms *name* and *name=*  
 95171 shall be equivalent to *name=""* for variables that take string values. The **set** command without  
 95172 arguments shall write a list of all defined variables and their values. The **no name** form shall be  
 95173 equivalent to **unset name**.

### 95174 **Invoke a Shell**

95175 *Synopsis:* `sh[ell]`

95176 Invoke an interactive command interpreter (see also *SHELL*).

### 95177 **Display Message Size**

95178 *Synopsis:* `si[ze] [msglist]`

95179 Write the size in bytes of each of the specified messages.

### 95180 **Read mailx Commands From a File**

95181 *Synopsis:* `so[urce] file`

95182 Read and execute commands from the file named by the pathname *file* and return to command  
 95183 mode.

### 95184 **Display Beginning of Messages**

95185 *Synopsis:* `to[p] [msglist]`

95186 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken  
 95187 as the number of lines to write. The default shall be 5.

### 95188 **Touch Messages**

95189 *Synopsis:* `to[uch] [msglist]`

95190 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a  
 95191 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

### 95192 **Delete Aliases**

95193 *Synopsis:* `una[lia] [alias]...`

95194 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

95195 **Undelete Messages**95196 *Synopsis:* `u[ndelete] [msglist]`95197 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last  
95198 message of those restored shall be written. If *msglist* is not specified, the message shall be  
95199 selected as follows:

- 95200 • If there are any deleted messages that follow the current message, the first of these shall be  
95201 chosen.
- 95202 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

95203 **Unset Variables**95204 *Synopsis:* `uns[et] name...`

95205 Cause the specified variables to be erased.

95206 **Edit Message with Full-Screen Editor**95207 *Synopsis:* `v[isual] [msglist]`95208 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,  
95209 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The  
95210 default editor shall be *vi*.95211 The **visual** command does not modify the contents of those messages in the mailbox.95212 **Write Messages to a File**95213 *Synopsis:* `w[rite] [msglist] file`95214 Write the given messages to the file specified by the pathname *file*, minus the message header.  
95215 Otherwise, it shall be equivalent to the **save** command.95216 **Scroll Header Display**95217 *Synopsis:* `z[+|-]`95218 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if  
95219 '-' is specified) one screenful. The number of headers written shall be set by the **screen**  
95220 variable.95221 **Invoke Shell Command**95222 *Synopsis:* `!command`95223 Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See  
95224 also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be  
95225 replaced with the command executed by the previous ! command or ~! command escape.

95226 **Null Command**95227 *Synopsis:* # *comment*95228 This null command (*comment*) shall be ignored by *mailx*.95229 **Display Current Message Number**95230 *Synopsis:* =

95231 Write the current message number.

95232 **Command Escapes in mailx**

95233 The following commands can be entered only from input mode, by beginning a line with the  
 95234 escape character (by default, <tilde> ('~')). See the **escape** variable description for changing  
 95235 this special character. The format for the commands shall be:

95236 *<escape-character><command-char><separator>[<argument(s)>]*95237 where the *<separator>* can be zero or more <blank> characters.

95238 In the following descriptions, the application shall ensure that the argument *command* (but not  
 95239 *mailx-command*) is a shell command string. Any string acceptable to the command interpreter  
 95240 specified by the *SHELL* variable when it is invoked as *SHELL -c command\_string* shall be valid.  
 95241 The command can be presented as multiple arguments (that is, quoting is not required).

95242 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send  
 95243 Mode and produce unspecified results.

95244 **~! *command*** Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and  
 95245 *command*; and then return to input mode. If the **bang** variable is set, each  
 95246 unescaped occurrence of '~!' in *command* shall be replaced with the command  
 95247 executed by the previous ! command or ~! command escape.

95248 **~.** Simulate end-of-file (terminate message input).

95249 **~: *mailx-command*, ~\_ *mailx-command***  
 95250 Perform the command-level request.

95251 **~?** Write a summary of command escapes.95252 **~A** This shall be equivalent to **~i Sign**.95253 **~a** This shall be equivalent to **~i sign**.95254 **~b *name...*** Add the *names* to the blind carbon copy (**Bcc**) list.95255 **~c *name...*** Add the *names* to the carbon copy (**Cc**) list.95256 **~d** Read in the dead-letter file. See *DEAD* for a description of this file.95257 **~e** Invoke the editor, as specified by the *EDITOR* environment variable, on the partial  
95258 message.

95259 **~f [*msglist*]** Forward the specified messages. The specified messages shall be inserted into the  
 95260 current message without alteration. This command escape also shall insert  
 95261 message headers into the message with field selection affected by the **discard**,  
 95262 **ignore**, and **retain** commands.

## mailx

## Utilities

- 95263           ~F [*msglist*] This shall be the equivalent of the ~f command escape, except that all headers shall  
95264           be included in the message, regardless of previous **discard**, **ignore**, and **retain**  
95265           commands.
- 95266           ~h           If standard input is a terminal, prompt for a **Subject** line and the **To**, **Cc**, and **Bcc**  
95267           lists. Other implementation-defined headers may also be presented for editing. If  
95268           the field is written with an initial value, it can be edited as if it had just been typed.
- 95269           ~i *string*    Insert the value of the named variable, followed by a <newline>, into the text of  
95270           the message. If the string is unset or null, the message shall not be changed.
- 95271           ~m [*msglist*] Insert the specified messages into the message, prefixing non-empty lines with the  
95272           string in the **indentprefix** variable. This command escape also shall insert message  
95273           headers into the message, with field selection affected by the **discard**, **ignore**, and  
95274           **retain** commands.
- 95275           ~M [*msglist*] This shall be the equivalent of the ~m command escape, except that all headers  
95276           shall be included in the message, regardless of previous **discard**, **ignore**, and **retain**  
95277           commands.
- 95278           ~p           Write the message being entered. If the message is longer than **crt** lines (see  
95279           [Internal Variables in mailx](#), on page 2889), the output shall be paginated as  
95280           described for the **PAGER** variable.
- 95281           ~q           Quit (see the **quit** command) from input mode by simulating an interrupt. If the  
95282           body of the message is not empty, the partial message shall be saved in the dead-  
95283           letter file. See **DEAD** for a description of this file.
- 95284           ~r *file*, ~< *file*, ~r !*command*, ~< !*command*  
95285           Read in the file specified by the pathname *file*. If the argument begins with an  
95286           <exclamation-mark> ('!'), the rest of the string shall be taken as an arbitrary  
95287           system command; the command interpreter specified by **SHELL** shall be invoked  
95288           with two arguments: **-c** and *command*. The standard output of *command* shall be  
95289           inserted into the message.
- 95290           ~s *string*    Set the subject line to *string*.
- 95291           ~t *name...*    Add the given *names* to the **To** list.
- 95292           ~v           Invoke the full-screen editor, as specified by the **VISUAL** environment variable, on  
95293           the partial message.
- 95294           ~w *file*       Write the partial message, without the header, onto the file named by the  
95295           pathname *file*. The file shall be created or the message shall be appended to it if  
95296           the file exists.
- 95297           ~x           Exit as with ~q, except the message shall not be saved in the dead-letter file.
- 95298           ~| *command*   Pipe the body of the message through the given *command* by invoking the  
95299           command interpreter specified by **SHELL** with two arguments: **-c** and *command*. If  
95300           the *command* returns a successful exit status, the standard output of the command  
95301           shall replace the message. Otherwise, the message shall remain unchanged. If the  
95302           *command* fails, an error message giving the exit status shall be written.

95303 **EXIT STATUS**95304 UP When the `-e` option is specified, the following exit values are returned:

95305 0 Mail was found.

95306 &gt;0 Mail was not found or an error occurred.

95307 Otherwise, the following exit values are returned:

95308 0 Successful completion; note that this status implies that all messages were *sent*, but it gives  
95309 no assurances that any of them were actually *delivered*.

95310 &gt;0 An error occurred.

95311 **CONSEQUENCES OF ERRORS**95312 UP When in `input mode (Receive Mode)` or `Send Mode`:95313 • If an error is encountered processing an input line beginning with a <tilde> ('~')  
95314 UP character, (see [Command Escapes in mailx](#), on page 2901), a diagnostic message shall be  
95315 written to standard error, and the message being composed may be modified, but this  
95316 condition shall not prevent the message from being sent.

95317 • Other errors shall prevent the sending of the message.

95318 UP When in `command mode`:

95319 • Default.

95320 **APPLICATION USAGE**95321 Delivery of messages to remote systems requires the existence of communication paths to such  
95322 systems. These need not exist.95323 Input lines are limited to {LINE\_MAX} bytes, but mailers between systems may impose more  
95324 severe line-length restrictions. This volume of POSIX.1-2008 does not place any restrictions on  
95325 the length of messages handled by *mailx*, and for delivery of local messages the only limitations  
95326 should be the normal problems of available disk space for the target mail file. When sending  
95327 messages to external machines, applications are advised to limit messages to less than 100 000  
95328 bytes because some mail gateways impose message-length restrictions.95329 The format of the system mailbox is intentionally unspecified. Not all systems implement  
95330 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some  
95331 system mailboxes may be multiple files, others records in a database. The internal format of the  
95332 messages themselves is specified with the historical format from Version 7, but only after the  
95333 messages have been saved in some file other than the system mailbox. This was done so that  
95334 many historical applications expecting text-file mailboxes are not broken.95335 Some new formats for messages can be expected in the future, probably including binary data,  
95336 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from  
95337 handling such messages, but it must store them as text files in secondary mailboxes (unless some  
95338 extension, such as a variable or command line option, is used to change the stored format). Its  
95339 method of doing so is implementation-defined and might include translating the data into text  
95340 file-compatible or readable form or omitting certain portions of the message from the stored  
95341 output.95342 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**  
95343 command discards all header-fields except those explicitly retained. The **discard** command  
95344 keeps all header-fields except those explicitly discarded. If headers exist on the retained header

95345 list, **discard** and **ignore** commands are ignored.

#### 95346 EXAMPLES

95347 None.

#### 95348 RATIONALE

95349 The standard developers felt strongly that a method for applications to send messages to specific  
95350 users was necessary. The obvious example is a batch utility, running non-interactively, that  
95351 wishes to communicate errors or results to a user. However, the actual format, delivery  
95352 mechanism, and method of reading the message are clearly beyond the scope of this volume of  
95353 POSIX.1-2008.

95354 The intent of this command is to provide a simple, portable interface for sending messages non-  
95355 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that  
95356 implementations explicitly denote the sender and recipient in the body of the delivered message.  
95357 Further specification of formats for either the message envelope or the message itself were  
95358 deliberately not made, as the industry is in the midst of changing from the current standards to a  
95359 more internationalized standard and it is probably incorrect, at this time, to require either one.

95360 Implementations are encouraged to conform to the various delivery mechanisms described in  
95361 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request  
95362 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

95363 Many historical systems modified each body line that started with **From** by prefixing the 'F'  
95364 with '>'. It is unnecessary, but allowed, to do that when the string does not follow a blank line  
95365 because it cannot be confused with the next header.

95366 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do  
95367 not modify the contents of those messages in the mailbox; such a capability could be added as an  
95368 extension, such as by using different command names.

95369 The restriction on a subject line being {LINE\_MAX}-10 bytes is based on the historical format  
95370 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a  
95371 message may encounter on other systems are not able to handle lines that long, however.

95372 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient  
95373 justification to exclude this utility from this volume of POSIX.1-2008. It is also arguable that it is,  
95374 in fact, testable, but that the tests themselves are not portable.

95375 When *mailx* is being used by an application that wishes to receive the results as if none of the  
95376 User Portability Utilities option features were supported, the *DEAD* environment variable must  
95377 be set to */dev/null*. Otherwise, it may be subject to the file creations described in *mailx*  
95378 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to  
95379 */dev/null*, historical versions of *mailx* and *Mail* read initialization commands from a file before  
95380 processing begins. Since the initialization that a user specifies could alter the contents of  
95381 messages an application is trying to send, such applications must set *MAILRC* to */dev/null*.

95382 The description of *LC\_TIME* uses “may affect” because many historical implementations do not  
95383 or cannot manipulate the date and time strings in the incoming mail headers. Some headers  
95384 found in incoming mail do not have enough information to determine the timezone in which the  
95385 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal  
95386 form that then is parsed by routines like *strftime()* that can take *LC\_TIME* settings into account.  
95387 Changing all these times to a user-specified format is allowed, but not required.

95388 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System  
95389 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by  
95390 declaring that *cat* is the paginator, would not meet with the intended meaning of this

95391 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her  
 95392 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike  
 95393 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common  
 95394 theme of user input, whereas editors differ dramatically.

95395 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to  
 95396 be format details and were omitted.

95397 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them  
 95398 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an  
 95399 appropriate marketing distinction between systems.

95400 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*  
 95401 option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain  
 95402 portable. However, it does force implementations to support usage such as:

```
95403 mailx -fin mymail.box
```

95404 The **no name** method of unsetting variables is not present in all historical systems, but it is in  
 95405 System V and provides a logical set of commands corresponding to the format of the display of  
 95406 options from the *mailx set* command without arguments.

95407 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the  
 95408 same feature. They are synonyms in this volume of POSIX 1-2008.

95409 The *mailx echo* command was not documented in the BSD version and has been omitted here  
 95410 because it is not obviously useful for interactive users.

95411 The default prompt on the System V *mailx* is a <question-mark>, on BSD *Mail* an <ampersand>.  
 95412 Since this volume of POSIX.1-2008 chose the *mailx* name, it kept the System V default, assuming  
 95413 that BSD users would not have difficulty with this minor incompatibility (that they can  
 95414 override).

95415 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,  
 95416 since this volume of POSIX.1-2008 chose the *mailx* name, it kept the System V default, but allows  
 95417 the SunOS user to achieve the desired results using **flipr**, an internal variable in System V *mailx*,  
 95418 although it has not been documented in the SVID.

95419 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command  
 95420 escapes were adopted from 4.3 BSD *Mail*.

95421 The **version** command was not included because no sufficiently general specification of the  
 95422 version information could be devised that would still be useful to a portable user. This  
 95423 command name should be used by suppliers who wish to provide version information about the  
 95424 *mailx* command.

95425 The “implementation-specific (unspecified) system start-up file” historically has been named  
 95426 **/etc/mailx.rc**, but this specific name and location are not required.

95427 The intent of the wording for the **next** command is that if any command has already displayed  
 95428 the current message it should display a following message, but, otherwise, it should display the  
 95429 current message. Consider the command sequence:

```
95430 next 3
95431 delete 3
95432 next
```

95433 where the **autoprint** option was not set. The normative text specifies that the second **next**  
 95434 command should display a message following the third message, because even though the

95435 current message has not been displayed since it was set by the **delete** command, it has been  
 95436 displayed since the current message was anything other than message number 3. This does not  
 95437 always match historical practice in some implementations, where the command file address  
 95438 followed by **next** (or the default command) would skip the message for which the user had  
 95439 searched.

#### 95440 FUTURE DIRECTIONS

95441 None.

#### 95442 SEE ALSO

95443 [Chapter 2](#) (on page 2297), *ed*, *ls*, *more*, *vi*

95444 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 95445 CHANGE HISTORY

95446 First released in Issue 2.

#### 95447 Issue 5

95448 The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default  
 95449 editor if this variable is not set. In previous issues, this default was not stated explicitly at this  
 95450 point but was implied further down in the text.

95451 The FUTURE DIRECTIONS section is added.

#### 95452 Issue 6

95453 The following new requirements on POSIX implementations derive from alignment with the  
 95454 Single UNIX Specification:

- 95455 • The **-F** option is added.
- 95456 • The **allnet**, **debug**, and **sendwait** internal variables are added.
- 95457 • The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

95458 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “*HOME*  
 95459 directory” is replaced by “directory referred to by the *HOME* environment variable”.

95460 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various  
 95461 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993  
 95462 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,  
 95463 #103, #106, #108, #114, #115, #122, and #129.

95464 The normative text is reworded to avoid use of the term “must” for application requirements.

95465 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

95466 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the  
 95467 EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was  
 95468 overlooked in the first version of this standard.

95469 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED  
 95470 DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from  
 95471 “implementation-defined” to “unspecified” for consistency with other text in the EXTENDED  
 95472 DESCRIPTION.

#### 95473 Issue 7

95474 Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC\_TIME*  
 95475 environment variable.

95476 Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next**  
 95477 command.

- 95478 SD5-XCU-ERN-69 is applied.
- 95479 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 95480 Shading to indicate support for the User Portability Utilities option is added.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**make**

Utilities

95481 **NAME**95482 **make** — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)95483 **SYNOPSIS**

```
95484 SD make [-einpqrst] [-f makefile]... [-k|-S] [macro=value...]
95485 [target_name...]
```

95486 **DESCRIPTION**

95487 The *make* utility shall update files that are derived from other files. A typical case is one where  
 95488 object files are derived from the corresponding source files. The *make* utility examines time  
 95489 relationships and shall update those derived files (called targets) that have modified times  
 95490 earlier than the modified times of the files (called prerequisites) from which they are derived. A  
 95491 description file (makefile) contains a description of the relationships between files, and the  
 95492 commands that need to be executed to update the targets to reflect changes in their  
 95493 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and  
 95494 optional commands to be executed when a prerequisite is newer than the target. There are two  
 95495 types of rule:

- 95496 1. *Inference rules*, which have one target name with at least one <period> ('.') and no  
 95497 <slash> ('/')
- 95498 2. *Target rules*, which can have more than one target name

95499 In addition, *make* shall have a collection of built-in macros and inference rules that infer  
 95500 prerequisite relationships to simplify maintenance of programs.

95501 To receive exactly the behavior described in this section, the user shall ensure that a portable  
 95502 makefile shall:

- 95503 • Include the special target **.POSIX**.
- 95504 • Omit any special target reserved for implementations (a leading period followed by  
 95505 uppercase letters) that has not been specified by this section

95506 The behavior of *make* is unspecified if either or both of these conditions are not met.

95507 **OPTIONS**

95508 The *make* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

95509 The following options shall be supported:

- 95510 **-e** Cause environment variables, including those with null values, to override macro  
 95511 assignments within makefiles.
- 95512 **-f *makefile*** Specify a different makefile. The argument *makefile* is a pathname of a description  
 95513 file, which is also referred to as the *makefile*. A pathname of '-' shall denote the  
 95514 standard input. There can be multiple instances of this option, and they shall be  
 95515 processed in the order specified. The effect of specifying the same option-argument  
 95516 more than once is unspecified.
- 95517 **-i** Ignore error codes returned by invoked commands. This mode is the same as if the  
 95518 special target **.IGNORE** were specified without prerequisites.
- 95519 **-k** Continue to update other targets that do not depend on the current target if a non-  
 95520 ignored error occurs while executing the commands to bring a target up-to-date.
- 95521 **-n** Write commands that would be executed on standard output, but do not execute  
 95522 them. However, lines with a <plus-sign> ('+') prefix shall be executed. In this  
 95523 mode, lines with an at-sign ('@') character prefix shall be written to standard

- 95524 output.
- 95525 **-p** Write to standard output the complete set of macro definitions and target  
95526 descriptions. The output format is unspecified.
- 95527 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit  
95528 value of 1. Targets shall not be updated if this option is specified. However, a  
95529 makefile command line (associated with the targets) with a <plus-sign> ('+')  
95530 prefix shall be executed.
- 95531 **-r** Clear the suffix list and do not use the built-in rules.
- 95532 **-S** Terminate *make* if an error occurs while executing the commands to bring a target  
95533 up-to-date. This shall be the default and the opposite of **-k**.
- 95534 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard  
95535 output before executing. This mode shall be the same as if the special target  
95536 **.SILENT** were specified without prerequisites.
- 95537 **-t** Update the modification time of each target as though a *touch target* had been  
95538 executed. Targets that have prerequisites but no commands (see **Target Rules**, on  
95539 page 2913), or that are already up-to-date, shall not be touched in this manner.  
95540 Write messages to standard output for each target file indicating the name of the  
95541 file and that it was touched. Normally, the *makefile* command lines associated with  
95542 each target are not executed. However, a command line with a <plus-sign> ('+')  
95543 prefix shall be executed.

95544 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any  
95545 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified  
95546 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option  
95547 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment  
95548 variable, the result is undefined.

#### 95549 OPERANDS

95550 The following operands shall be supported:

95551 *target\_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is  
95552 specified, while *make* is processing the makefiles, the first target that *make*  
95553 encounters that is not a special target or an inference rule shall be used.

95554 *macro=value* Macro definitions, as defined in **Macros** (on page 2914).

95555 If the *target\_name* and *macro=value* operands are intermixed on the *make* utility command line,  
95556 the results are unspecified.

#### 95557 STDIN

95558 The standard input shall be used only if the *makefile* option-argument is **'-'**. See the INPUT  
95559 FILES section.

#### 95560 INPUT FILES

95561 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,  
95562 and comments. See the EXTENDED DESCRIPTION section.

#### 95563 ENVIRONMENT VARIABLES

95564 The following environment variables shall affect the execution of *make*:

95565 **LANG** Provide a default value for the internationalization variables that are unset or null.  
95566 (See XBD **Section 8.2** (on page 174) for the precedence of internationalization  
95567 variables used to determine the values of locale categories.)

|       |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|-----|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 95568 |     | <i>LC_ALL</i>              | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 95569 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95570 |     | <i>LC_CTYPE</i>            | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 95571 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95572 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95573 |     | <i>LC_MESSAGES</i>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95574 |     |                            | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 95575 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95576 |     | <i>MAKEFLAGS</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95577 |     |                            | This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):                                                                                                                                                                                                                                                                                                                                                                                     |
| 95578 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95579 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95580 |     |                            | <ul style="list-style-type: none"> <li>• The characters are option letters without the leading &lt;hyphen&gt; characters or &lt;blank&gt; separation used on a <i>make</i> utility command line.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 95581 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95582 |     |                            | <ul style="list-style-type: none"> <li>• The characters are formatted in a manner similar to a portion of the <i>make</i> utility command line: options are preceded by &lt;hyphen&gt; characters and &lt;blank&gt;-separated as described in XBD Section 12.2 (on page 215). The <i>macro=value</i> macro definition operands can also be included. The difference between the contents of <i>MAKEFLAGS</i> and the <i>make</i> utility command line is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6, on page 2305) associated with parsing the command line values.</li> </ul> |
| 95583 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95584 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95585 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95586 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95587 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95588 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95589 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95590 | XSI | <i>NLSPATH</i>             | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 95591 | XSI | <i>PROJECTDIR</i>          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95592 |     |                            | Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory <b>SCCS</b> in the identified directory. If the value of <i>PROJECTDIR</i> begins with a <slash>, it shall be considered an absolute pathname; otherwise, the value of <i>PROJECTDIR</i> is treated as a user name and that user's initial working directory shall be examined for a subdirectory <b>src</b> or <b>source</b> . If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.                   |
| 95593 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95594 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95595 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95596 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95597 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95598 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95599 |     |                            | If <i>PROJECTDIR</i> is not set or has a null value, the search for SCCS files shall be made in the directory <b>SCCS</b> in the current directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 95600 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95601 |     |                            | The setting of <i>PROJECTDIR</i> affects all files listed in the remainder of this utility description for files with a component named <b>SCCS</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 95602 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95603 |     |                            | The value of the <i>SHELL</i> environment variable shall not be used as a macro and shall not be modified by defining the <b>SHELL</b> macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in <a href="#">Macros</a> (on page 2914).                                                                                                                                                                                                                                                                                                  |
| 95604 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95605 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95606 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95607 |     | <b>ASYNCHRONOUS EVENTS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95608 |     |                            | If not already ignored, <i>make</i> shall trap <b>SIGHUP</b> , <b>SIGTERM</b> , <b>SIGINT</b> , and <b>SIGQUIT</b> and remove the current target unless the target is a directory or the target is a prerequisite of the special target <b>.PRECIOUS</b> or unless one of the <b>-n</b> , <b>-p</b> , or <b>-q</b> options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, <i>make</i> shall take the standard action for all other signals.                                                        |
| 95609 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95610 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95611 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95612 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 95613 |     |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

95614 **STDOUT**

95615 The *make* utility shall write all commands to be executed to standard output unless the `-s` option  
 95616 was specified, the command is prefixed with an at-sign, or the special target **.SILENT** has either  
 95617 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work  
 95618 needing to be done, it shall write a message to standard output indicating that no action was  
 95619 taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a  
 95620 message of unspecified format indicating that the file was touched, including the filename of the  
 95621 file.

95622 **STDERR**

95623 The standard error shall be used only for diagnostic messages.

95624 **OUTPUT FILES**

95625 Files can be created when the `-t` option is present. Additional files can also be created by the  
 95626 utilities invoked by *make*.

95627 **EXTENDED DESCRIPTION**

95628 The *make* utility attempts to perform the actions required to ensure that the specified targets are  
 95629 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it  
 95630 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively  
 95631 ensure that they are up-to-date, processing them in the order in which they appear in the rule.  
 95632 The *make* utility shall use the modification times of files to determine whether the corresponding  
 95633 targets are out-of-date.

95634 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is  
 95635 out-of-date, the commands associated with the target entry shall be executed. If there are no  
 95636 commands listed for the target, the target shall be treated as up-to-date.

95637 **Makefile Syntax**

95638 A makefile can contain rules, macro definitions (see [Macros](#), on page 2914), include lines, and  
 95639 comments. There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall  
 95640 contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be  
 95641 used and the suffix list shall be cleared. Additional rules of both types can be specified in a  
 95642 makefile. If a rule is defined more than once, the value of the rule shall be that of the last one  
 95643 specified. Macros can also be defined more than once, and the value of the macro is specified in  
 95644 [Macros](#) (on page 2914). Comments start with a <number-sign> ('#') and continue until an  
 95645 unescaped <newline> is reached.

95646 By default, the following files shall be tried in sequence: **.makefile** and **.Makefile**. If neither  
 95647 **.makefile** or **.Makefile** are found, other implementation-defined files may also be tried. On  
 95648 XSI-conformant systems, the additional files **.s.makefile**, **SCCS/s.makefile**, **.s.Makefile**, and  
 95649 **SCCS/s.Makefile** shall also be tried.

95650 The `-f` option shall direct *make* to ignore any of these default files and use the specified argument  
 95651 as a makefile instead. If the `'-'` argument is specified, standard input shall be used.

95652 The term *makefile* is used to refer to any rules provided by the user, whether in **.makefile** or its  
 95653 variants, or specified by the `-f` option.

95654 The rules in makefiles shall consist of the following types of lines: target rules, including special  
 95655 targets (see [Target Rules](#), on page 2913), inference rules (see [Inference Rules](#), on page 2916),  
 95656 macro definitions (see [Macros](#), on page 2914), empty lines, and comments.

95657 Target and Inference Rules may contain *command lines*. Command lines can have a prefix that  
 95658 shall be removed before execution (see [Makefile Execution](#), on page 2912).

95659 When an escaped <newline> (one preceded by a <backslash>) is found anywhere in the  
 95660 makefile except in a command line, an include line, or a line immediately preceding an include  
 95661 line, it shall be replaced, along with any leading white space on the following line, with a single  
 95662 <space>. When an escaped <newline> is found in a command line in a makefile, the command  
 95663 line shall contain the <backslash>, the <newline>, and the next line, except that the first  
 95664 character of the next line shall not be included if it is a <tab>. When an escaped <newline> is  
 95665 found in an include line or in a line immediately preceding an include line, the behavior is  
 95666 unspecified.

#### 95667 **Include Lines**

95668 If the word **include** appears at the beginning of a line and is followed by one or more <blank>  
 95669 characters, the string formed by the remainder of the line shall be processed as follows to  
 95670 produce a pathname:

- 95671 • The trailing <newline> and any comment shall be discarded. If the resulting string  
 95672 contains any double-quote characters ( ' " ' ) the behavior is unspecified.
- 95673 • The resulting string shall be processed for macro expansion (see **Macros** (on page 2914).
- 95674 • Any <blank> characters that appear after the first non-<blank> shall be used as separators  
 95675 to divide the macro-expanded string into fields. It is unspecified whether any other white-  
 95676 space characters are also used as separators. It is unspecified whether pathname expansion  
 95677 (see **Section 2.13**, on page 2332) is also performed.
- 95678 • If the processing of separators and optional pathname expansion results in either zero or  
 95679 two or more non-empty fields, the behavior is unspecified. If it results in one non-empty  
 95680 field, that field is taken as the pathname.

95681 If the pathname does not begin with a '/' it shall be treated as relative to the current working  
 95682 directory of the process, not relative to the directory containing the makefile. If the file does not  
 95683 exist in this location, it is unspecified whether additional directories are searched.

95684 The contents of the file specified by the pathname shall be read and processed as if they  
 95685 appeared in the makefile in place of the include line. If the file ends with an escaped <newline>  
 95686 the behavior is unspecified.

95687 The file may itself contain further include lines. Implementations shall support nesting of  
 95688 include files up to a depth of at least 16.

#### 95689 **Makefile Execution**

95690 Makefile command lines shall be processed one at a time.

95691 Makefile command lines can have one or more of the following prefixes: a <hyphen> ('-'), an  
 95692 at-sign ('@'), or a <plus-sign> ('+'). These shall modify the way in which *make* processes the  
 95693 command.

- 95694 – If the command prefix contains a <hyphen>, or the **-i** option is present, or the special target  
 95695 **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error  
 95696 found while executing the command shall be ignored.
- 95697 @ If the command prefix contains an at-sign and the *make* utility command line **-n** option is  
 95698 not specified, or the **-s** option is present, or the special target **.SILENT** has either the current  
 95699 target as a prerequisite or has no prerequisites, the command shall not be written to  
 95700 standard output before it is executed.

95701 + If the command prefix contains a <plus-sign>, this indicates a makefile command line that  
95702 shall be executed even if **-n**, **-q**, or **-t** is specified.

95703 An *execution line* is built from the command line by removing any prefix characters. Except as  
95704 described under the at-sign prefix, the execution line shall be written to the standard output,  
95705 optionally preceded by a <tab>. The execution line shall then be executed by a shell as if it were  
95706 passed as the argument to the *system()* interface, except that the shell **-e** option shall also be in  
95707 effect. The environment for the command being executed shall contain all of the variables in the  
95708 environment of *make*.

95709 By default, when *make* receives a non-zero status from the execution of a command, it shall  
95710 terminate with an error message to standard error.

### 95711 Target Rules

95712 Target rules are formatted as follows:

```
95713 target [target...]: [prerequisite...] [; command]
95714 [command
95715 <tab>command
95716 ...]
```

95717 *line that does not begin with <tab>*

95718 Target entries are specified by a <blank>-separated, non-null list of targets, then a <colon>, then  
95719 a <blank>-separated, possibly empty list of prerequisites. Text following a <semicolon>, if any,  
95720 and all following lines that begin with a <tab>, are makefile command lines to be executed to  
95721 update the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a  
95722 new entry. An empty or blank line, or a line beginning with '#', may begin a new entry.

95723 Applications shall select target names from the set of characters consisting solely of periods,  
95724 underscores, digits, and alphabets from the portable character set (see XBD Section 6.1, on  
95725 page 125). Implementations may allow other characters in target names as extensions. The  
95726 interpretation of targets containing the characters '%' and '"' is implementation-defined.

95727 A target that has prerequisites, but does not have any commands, can be used to add to the  
95728 prerequisite list for that target. Only one target rule for any given target can contain commands.

95729 Lines that begin with one of the following are called *special targets* and control the operation of  
95730 *make*:

95731 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is  
95732 specified with commands, but without prerequisites. The commands shall be used  
95733 by *make* if there are no other rules available to build a target.

95734 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors  
95735 from commands associated with them to be ignored in the same manner as  
95736 specified by the **-i** option. Subsequent occurrences of **.IGNORE** shall add to the  
95737 list of targets ignoring command errors. If no prerequisites are specified, *make* shall  
95738 behave as if the **-i** option had been specified and errors from all commands  
95739 associated with all targets shall be ignored.

95740 **.POSIX** The application shall ensure that this special target is specified without  
95741 prerequisites or commands. If it appears as the first non-comment line in the  
95742 makefile, *make* shall process the makefile as specified by this section; otherwise, the  
95743 behavior of *make* is unspecified.

95744 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the  
 95745 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS  
 95746 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious  
 95747 files. If no prerequisites are specified, all targets in the makefile shall be treated as  
 95748 if specified with **.PRECIOUS**.

95749 XSI **.SCCS\_GET** The application shall ensure that this special target is specified without  
 95750 prerequisites. If this special target is included in a makefile, the commands  
 95751 specified with this target shall replace the default commands associated with this  
 95752 special target (see [Default Rules](#), on page 2919). The commands specified with this  
 95753 target are used to get all SCCS files that are not found in the current directory.

95754 When source files are named in a dependency list, *make* shall treat them just like  
 95755 any other target. Because the source file is presumed to be present in the directory,  
 95756 there is no need to add an entry for it to the makefile. When a target has no  
 95757 dependencies, but is present in the directory, *make* shall assume that that file is up-  
 95758 to-date. If, however, an SCCS file named **SCCS/s.source\_file** is found for a target  
 95759 *source\_file*, *make* compares the timestamp of the target file with that of the  
 95760 **SCCS/s.source\_file** to ensure the target is up-to-date. If the target is missing, or if  
 95761 the SCCS file is newer, *make* shall automatically issue the commands specified for  
 95762 the **.SCCS\_GET** special target to retrieve the most recent version. However, if the  
 95763 target is writable by anyone, *make* shall not retrieve a new version.

95764 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause  
 95765 commands associated with them not to be written to the standard output before  
 95766 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of  
 95767 targets with silent commands. If no prerequisites are specified, *make* shall behave  
 95768 as if the **-s** option had been specified and no commands or touch messages  
 95769 associated with any target shall be written to standard output.

95770 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are  
 95771 used in conjunction with the inference rules (see [Inference Rules](#), on page 2916). If  
 95772 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be  
 95773 cleared.

95774 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified  
 95775 without commands.

95776 Targets with names consisting of a leading <period> followed by the uppercase letters "POSIX"  
 95777 and then any other characters are reserved for future standardization. Targets with names  
 95778 consisting of a leading <period> followed by one or more uppercase letters are reserved for  
 95779 implementation extensions.

## 95780 **Macros**

95781 Macro definitions are in the form:

```
95782 string1 = [string2]
```

95783 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all  
 95784 characters, if any, after the <equals-sign>, up to a comment character ('#') or an unescaped  
 95785 <newline>. Any <blank> characters immediately before or after the <equals-sign> shall be  
 95786 ignored.

95787 Applications shall select macro names from the set of characters consisting solely of periods,  
 95788 underscores, digits, and alphabetic characters from the portable character set (see [XBD Section 6.1](#), on  
 95789 page 125). A macro name shall not contain an <equals-sign>. Implementations may allow other

95790 characters in macro names as extensions.

95791 Macros can appear anywhere in the makefile. Macro expansions using the forms  $\$(string1)$  or  
95792  $\${string1}$  shall be replaced by  $string2$ , as follows:

- 95793 • Macros in target lines shall be evaluated when the target line is read.
- 95794 • Macros in makefile command lines shall be evaluated when the command is executed.
- 95795 • Macros in the string before the <equals-sign> in a macro definition shall be evaluated  
95796 when the macro assignment is made.
- 95797 • Macros after the <equals-sign> in a macro definition shall not be evaluated until the  
95798 defined macro is used in a rule or command, or before the <equals-sign> in a macro  
95799 definition.

95800 The parentheses or braces are optional if  $string1$  is a single character. The macro  $\$\$$  shall be  
95801 replaced by the single character '\$'. If  $string1$  in a macro expansion contains a macro  
95802 expansion, the results are unspecified.

95803 Macro expansions using the forms  $\$(string1[:subst1=[subst2]])$  or  $\${string1[:subst1=[subst2]]}$  can  
95804 be used to replace all occurrences of  $subst1$  with  $subst2$  when the macro substitution is  
95805 performed. The  $subst1$  to be replaced shall be recognized when it is a suffix at the end of a word  
95806 in  $string1$  (where a *word*, in this context, is defined to be a string delimited by the beginning of  
95807 the line, a <blank>, or a <newline>). If  $string1$  in a macro expansion contains a macro expansion,  
95808 the results are unspecified.

95809 Macro expansions in  $string1$  of macro definition lines shall be evaluated when read. Macro  
95810 expansions in  $string2$  of macro definition lines shall be performed when the macro identified by  
95811  $string1$  is expanded in a rule or command.

95812 Macro definitions shall be taken from the following sources, in the following logical order,  
95813 before the makefile(s) are read.

- 95814 1. Macros specified on the *make* utility command line, in the order specified on the  
95815 command line. It is unspecified whether the internal macros defined in [Internal Macros](#)  
95816 (on page 2917) are accepted from this source.
- 95817 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the  
95818 environment variable. It is unspecified whether the internal macros defined in [Internal](#)  
95819 [Macros](#) (on page 2917) are accepted from this source.
- 95820 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and  
95821 including the variables with null values.
- 95822 4. Macros defined in the inference rules built into *make*.

95823 Macro definitions from these sources shall not override macro definitions from a lower-  
95824 numbered source. Macro definitions from a single source (for example, the *make* utility  
95825 command line, the *MAKEFLAGS* environment variable, or the other environment variables)  
95826 shall override previous macro definitions from the same source.

95827 Macros defined in the makefile(s) shall override macro definitions that occur before them in the  
95828 makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined  
95829 in the makefile(s) shall override macro definitions from source 3. Macros defined in the  
95830 makefile(s) shall not override macro definitions from source 1 or source 2.

95831 Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*)  
95832 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not  
95833 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted

95834 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance  
 95835 of the *make* command, the original macro's value is recovered. Other implementation-defined  
 95836 options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of  
 95837 the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the  
 95838 *MAKEFLAGS* environment variable shall be modified to match the new value of the  
 95839 *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

95840 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the  
 95841 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other  
 95842 implementation-defined variables may also be added to the environment of *make*.

95843 The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the  
 95844 pathname of the shell command language interpreter (see *sh*). The *SHELL* environment variable  
 95845 shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified  
 95846 on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect  
 95847 the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the  
 95848 command line are implementation-defined.

#### 95849 Inference Rules

95850 Inference rules are formatted as follows:

```
95851 target:
95852 <tab>command
95853 [<tab>command]
95854 ...
95855 line that does not begin with <tab> or #
```

95856 The application shall ensure that the *target* portion is a valid target name (see [Target Rules](#), on  
 95857 page 2913) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as  
 95858 prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any *<slash>* or  
 95859 *<period>* characters.) If there is only one *<period>* in the target, it is a single-suffix inference  
 95860 rule. Targets with two periods are double-suffix inference rules. Inference rules can have only  
 95861 one target before the *<colon>*.

95862 The application shall ensure that the makefile does not specify prerequisites for inference rules;  
 95863 no characters other than white space shall follow the *<colon>* in the first line, except when  
 95864 creating the *empty rule*, described below. Prerequisites are inferred, as described below.

95865 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite  
 95866 the old inference rule. An empty rule can be created with a command consisting of simply a  
 95867 *<semicolon>* (that is, the rule still exists and is found during inference rule search, but since it is  
 95868 empty, execution has no effect). The empty rule can also be formatted as follows:

```
95869 rule: ;
```

95870 where zero or more *<blank>* characters separate the *<colon>* and *<semicolon>*.

95871 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be  
 95872 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*  
 95873 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

95874 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by  
 95875 the inference rules. The order in which the suffixes are specified defines the order in which the  
 95876 inference rules for the suffixes are used. New suffixes shall be appended to the current list by  
 95877 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites  
 95878 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is

95879 required to change the order of the suffixes.

95880 Normally, the user would provide an inference rule for each suffix. The inference rule to update  
 95881 a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The  
 95882 internal macros provide the means to specify general inference rules (see [Internal Macros](#)).

95883 When no target rule is found to update a target, the inference rules shall be checked. The suffix  
 95884 of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special  
 95885 targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order  
 95886 defined for the first **.s2.s1** rule whose prerequisite file (**\$.s2**) exists. If the target is out-of-date  
 95887 with respect to this prerequisite, the commands for that inference rule shall be executed.

95888 If the target to be built does not contain a suffix and there is no rule for the target, the single  
 95889 suffix inference rules shall be checked. The single-suffix inference rules define how to build a  
 95890 target if a file is found with a name that matches the target name with one of the single suffixes  
 95891 appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The  
 95892 other suffix (**.s1**) is treated as null.

95893 XSI A `<tilde>` ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule  
 95894 **.c~.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of  
 95895 the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the '~' is a  
 95896 way of changing any file reference into an SCCS file reference.

### 95897 Libraries

95898 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive  
 95899 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*  
 95900 to the member name. The application shall ensure that the member is an object file with the **.o**  
 95901 suffix. The modification time of the expression is the modification time for the member as kept  
 95902 in the archive library; see *ar*. The **.a** suffix shall refer to an archive library. The **.s2.a** rule shall be  
 95903 used to update a member in the library from a file with a suffix **.s2**.

### 95904 Internal Macros

95905 The *make* utility shall maintain five internal macros that can be used in target and inference rules.  
 95906 In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,  
 95907 *inference rule*, *target*, and *prerequisite* is necessary.

95908 Target rules are specified by the user in a makefile for a particular target. Inference rules are  
 95909 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites  
 95910 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those  
 95911 prerequisites that are generated when inference rules are used. Inference rules are applied to  
 95912 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in  
 95913 the makefile. Target rules are applied to targets specified in the makefile.

95914 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)  
 95915 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon  
 95916 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be  
 95917 processed recursively until a target is found that has no prerequisites, at which point the  
 95918 recursion stops. The recursion shall then back up, updating each target as it goes.

95919 In the definitions that follow, the word *target* refers to one of:

- 95920 • A target specified in the makefile

95921 • An explicit prerequisite specified in the makefile that becomes the target when *make*  
95922 processes it during recursion

95923 • An implicit prerequisite that becomes a target when *make* processes it during recursion

95924 In the definitions that follow, the word *prerequisite* refers to one of the following:

95925 • An explicit prerequisite specified in the makefile for a particular target

95926 • An implicit prerequisite generated as a result of locating an appropriate inference rule and  
95927 corresponding file that matches the suffix of the target

95928 The five internal macros are:

95929 **\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive  
95930 filename part of a library archive target. It shall be evaluated for both target and  
95931 inference rules.

95932 For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.  
95933 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-  
95934 date **lib.a**.

95935 **\$\$** The **\$\$** macro shall be evaluated only when the current target is an archive library  
95936 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and  
95937 **\$\$** shall evaluate to *member.o*. The **\$\$** macro shall be evaluated for both target and  
95938 inference rules.

95939 For example, in a makefile target rule to build **lib.a(file.o)**, **\$\$** represents **file.o**, as  
95940 opposed to **\$@**, which represents **lib.a**.

95941 **\$\$?** The **\$\$?** macro shall evaluate to the list of prerequisites that are newer than the current  
95942 target. It shall be evaluated for both target and inference rules.

95943 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and  
95944 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to  
95945 **file2.o** and **file3.o**, **\$\$?** represents **file2.o** and **file3.o**.

95946 **\$\$<** In an inference rule, the **\$\$<** macro shall evaluate to the filename whose existence  
95947 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the **\$\$<**  
95948 macro shall evaluate to the current target name. The meaning of the **\$\$<** macro shall be  
95949 otherwise unspecified.

95950 For example, in the **.c.a** inference rule, **\$\$<** represents the prerequisite **.c** file.

95951 **\$\$\*** The **\$\$\*** macro shall evaluate to the current target name with its suffix deleted. It shall be  
95952 evaluated at least for inference rules.

95953 For example, in the **.c.a** inference rule, **\$\$\*.o** represents the out-of-date **.o** file that  
95954 corresponds to the prerequisite **.c** file.

95955 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended  
95956 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*  
95957 for 'F'. The directory part is the path prefix of the file without a trailing <slash>; for the current  
95958 directory, the directory part is '.'. When the **\$\$?** macro contains more than one prerequisite  
95959 filename, the **\$\$(?D)** and **\$\$(?F)** (or **\$\${?D}** and **\$\${?F}**) macros expand to a list of directory name parts  
95960 and filename parts respectively.

95961 For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as:

## Utilities

## make

95962        \$<     *member.s2*  
 95963        \$\*     *member*  
 95964        \$@     *lib*  
 95965        \$?     *member.s2*  
 95966        \$%     *member.o*

95967        **Default Rules**

95968        The default rules for *make* shall achieve results that are the same as if the following were used.  
 95969        Implementations that do not support the C-Language Development Utilities option may omit  
 95970        **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.  
 95971        Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference  
 95972        rules. Implementations may provide additional macros and rules.

95973        *SPECIAL TARGETS*

95974    XSI     **.SCCS\_GET:** *sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@*

95975    XSI     **.SUFFIXES:** *.o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~*

95976        **MACROS**

95977        MAKE=make  
 95978        AR=ar  
 95979        ARFLAGS=-rv  
 95980        YACC=yacc  
 95981        YFLAGS=  
 95982        LEX=lex  
 95983        LFLAGS=  
 95984        LDFLAGS=  
 95985        CC=c99  
 95986        CFLAGS=-O  
 95987        FC=fort77  
 95988        FFLAGS=-O 1

95989    XSI     **GET=get**  
 95990        **GFLAGS=**  
 95991        **SCCSFLAGS=**  
 95992        **SCCSGETFLAGS=-s**

95993        *SINGLE SUFFIX RULES*

95994        **.c:**  
 95995        \$ (CC) \$ (CFLAGS) \$ (LDFLAGS) -o \$@ \$<

95996        **.f:**  
 95997        \$ (FC) \$ (FFLAGS) \$ (LDFLAGS) -o \$@ \$<

95998        **.sh:**  
 95999        cp \$< \$@  
 96000        chmod a+x \$@

96001    XSI     **.c~:**  
 96002        \$ (GET) \$ (GFLAGS) -p \$< > \$\*.c

## make

## Utilities

```

96003      $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
96004 .f~:
96005      $(GET) $(GFLAGS) -p $< > $*.f
96006      $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f

96007 .sh~:
96008      $(GET) $(GFLAGS) -p $< > $*.sh
96009      cp $*.sh $@
96010      chmod a+x $@

```

96011 *DOUBLE SUFFIX RULES*

```

96012 .c.o:
96013      $(CC) $(CFLAGS) -c $<

96014 .f.o:
96015      $(FC) $(FFLAGS) -c $<

96016 .y.o:
96017      $(YACC) $(YFLAGS) $<
96018      $(CC) $(CFLAGS) -c y.tab.c
96019      rm -f y.tab.c
96020      mv y.tab.o $@

96021 .l.o:
96022      $(LEX) $(LFLAGS) $<
96023      $(CC) $(CFLAGS) -c lex.yy.c
96024      rm -f lex.yy.c
96025      mv lex.yy.o $@

96026 .y.c:
96027      $(YACC) $(YFLAGS) $<
96028      mv y.tab.c $@

96029 .l.c:
96030      $(LEX) $(LFLAGS) $<
96031      mv lex.yy.c $@

```

```

96032 XSI .c~.o:
96033      $(GET) $(GFLAGS) -p $< > $*.c
96034      $(CC) $(CFLAGS) -c $*.c

96035 .f~.o:
96036      $(GET) $(GFLAGS) -p $< > $*.f
96037      $(FC) $(FFLAGS) -c $*.f

96038 .y~.o:
96039      $(GET) $(GFLAGS) -p $< > $*.y
96040      $(YACC) $(YFLAGS) $*.y
96041      $(CC) $(CFLAGS) -c y.tab.c
96042      rm -f y.tab.c
96043      mv y.tab.o $@

96044 .l~.o:
96045      $(GET) $(GFLAGS) -p $< > $*.l
96046      $(LEX) $(LFLAGS) $*.l

```

```

96047     $(CC) $(CFLAGS) -c lex.yy.c
96048     rm -f lex.yy.c
96049     mv lex.yy.o $@

96050 .y~.c:
96051     $(GET) $(GFLAGS) -p $< > $*.y
96052     $(YACC) $(YFLAGS) $*.y
96053     mv y.tab.c $@

96054 .l~.c:
96055     $(GET) $(GFLAGS) -p $< > $*.l
96056     $(LEX) $(LFLAGS) $*.l
96057     mv lex.yy.c $@

```

```

96058 .c.a:
96059     $(CC) -c $(CFLAGS) $<
96060     $(AR) $(ARFLAGS) $@ $*.o
96061     rm -f $*.o

96062 .f.a:
96063     $(FC) -c $(FFLAGS) $<
96064     $(AR) $(ARFLAGS) $@ $*.o
96065     rm -f $*.o

```

#### 96066 EXIT STATUS

96067 When the `-q` option is specified, the *make* utility shall exit with one of the following values:

- 96068 0 Successful completion.
- 96069 1 The target was not up-to-date.
- 96070 >1 An error occurred.

96071 When the `-q` option is not specified, the *make* utility shall exit with one of the following values:

- 96072 0 Successful completion.
- 96073 >0 An error occurred.

#### 96074 CONSEQUENCES OF ERRORS

96075 Default.

#### 96076 APPLICATION USAGE

96077 If there is a source file (such as `./source.c`) and there are two SCCS files corresponding to it  
 96078 (`./s.source.c` and `./SCCS/s.source.c`), on XSI-conformant systems *make* uses the SCCS file in the  
 96079 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,  
 96080 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for  
 96081 a given source file, future developers are very likely to be confused.

96082 It is incumbent upon portable makefiles to specify the `.POSIX` special target in order to  
 96083 guarantee that they are not affected by local extensions.

96084 The `-k` and `-S` options are both present so that the relationship between the command line, the  
 96085 `MAKEFLAGS` variable, and the makefile can be controlled precisely. If the `k` flag is passed in  
 96086 `MAKEFLAGS` and a command is of the form:

```
96087 $(MAKE) -S foo
```

96088 then the default behavior is restored for the child *make*.

96089 When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive  
96090 `make -n target` to be used to see all of the action that would be taken to update `target`.

96091 Because of widespread historical practice, interpreting a `<number-sign>` (`'#'`) inside a variable  
96092 as the start of a comment has the unfortunate side-effect of making it impossible to place a  
96093 `<number-sign>` in a variable, thus forbidding something like:

```
96094 CFLAGS = "-D COMMENT_CHAR='#'"
```

96095 Many historical `make` utilities stop chaining together inference rules when an intermediate target  
96096 is nonexistent. For example, it might be possible for a `make` to determine that both `.y.c` and `.c.o`  
96097 could be used to convert a `.y` to a `.o`. Instead, in this case, `make` requires the use of a `.y.o` rule.

96098 The best way to provide portable makefiles is to include all of the rules needed in the makefile  
96099 itself. The rules provided use only features provided by other parts of this volume of  
96100 POSIX.1-2008. The default rules include rules for optional commands in this volume of  
96101 POSIX.1-2008. Only rules pertaining to commands that are provided are needed in an  
96102 implementation's default set.

96103 Macros used within other macros are evaluated when the new macro is used rather than when  
96104 the new macro is defined. Therefore:

```
96105 MACRO = value1
96106 NEW = $(MACRO)
96107 MACRO = value2
```

```
96108 target:
96109     echo $(NEW)
```

96110 would produce `value2` and not `value1` since `NEW` was not expanded until it was needed in the  
96111 `echo` command line.

96112 Some historical applications have been known to intermix `target_name` and `macro=name` operands  
96113 on the command line, expecting that all of the macros are processed before any of the targets are  
96114 dealt with. Conforming applications do not do this, although some backwards-compatibility  
96115 support may be included in some implementations.

96116 The following characters in filenames may give trouble: `'='`, `':'`, `'\'`, single-quote, and `'@'`.  
96117 In include filenames, pattern matching characters and `'\"'` should also be avoided, as they may  
96118 be treated as special by some implementations.

96119 For inference rules, the description of `$<` and `$?` seem similar. However, an example shows the  
96120 minor difference. In a makefile containing:

```
96121 foo.o: foo.h
```

96122 if `foo.h` is newer than `foo.o`, yet `foo.c` is older than `foo.o`, the built-in rule to make `foo.o` from  
96123 `foo.c` is used, with `$<` equal to `foo.c` and `$?` equal to `foo.h`. If `foo.c` is also newer than `foo.o`, `$<` is  
96124 equal to `foo.c` and `$?` is equal to `foo.h`.

## 96125 EXAMPLES

- 96126 1. The following command:  
96127 `make`  
96128 makes the first target found in the makefile.
- 96129 2. The following command:  
96130 `make junk`

- 96131 makes the target **junk**.
- 96132 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in  
96133 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:
- ```
96134 pgm: a.o b.o
96135     c99 a.o b.o -o pgm
96136 a.o: incl.h a.c
96137     c99 -c a.c
96138 b.o: incl.h b.c
96139     c99 -c b.c
```
- 96140 4. An example for making optimized **.o** files from **.c** files is:
- ```
96141 .c.o:
96142     c99 -c -O $*.c
```
- 96143 or:
- ```
96144 .c.o:
96145     c99 -c -O $<
```
- 96146 5. The most common use of the archive interface follows. Here, it is assumed that the source  
96147 files are all C-language source:
- ```
96148 lib: lib(file1.o) lib(file2.o) lib(file3.o)
96149     @echo lib is now up-to-date
```
- 96150 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.
- 96151 The treatment of escaped <newline> characters throughout the makefile is historical  
96152 practice. For example, the inference rule:
- ```
96153 .c.o\
96154 :
```
- 96155 works, and the macro:
- ```
96156 f= bar baz\
96157     biz
96158 a:
96159     echo ==$f==
```
- 96160 echoes "==bar baz biz==".
- 96161 If **?** were:
- ```
96162 /usr/include/stdio.h /usr/include/unistd.h foo.h
```
- 96163 then **\$(?D)** would be:
- ```
96164 /usr/include /usr/include .
```
- 96165 and **\$(?F)** would be:
- ```
96166 stdio.h unistd.h foo.h
```
- 96167 6. The contents of the built-in rules can be viewed by running:
- ```
96168 make -p -f /dev/null 2>/dev/null
```

## 96169 RATIONALE

96170 The *make* utility described in this volume of POSIX.1-2008 is intended to provide the means for  
 96171 changing portable source code into executables that can be run on an POSIX.1-2008-conforming  
 96172 system. It reflects the most common features present in System V and BSD *makes*.

96173 Historically, the *make* utility has been an especially fertile ground for vendor and research  
 96174 organization-specific syntax modifications and extensions. Examples include:

- 96175 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,  
 96176 and others)
- 96177 • Additional “operators” separating targets and their prerequisites (System V, BSD, and  
 96178 others)
- 96179 • Specifying that command lines containing the strings “\${MAKE}” and “\$(MAKE)” are  
 96180 executed when the `-n` option is specified (GNU and System V)
- 96181 • Modifications of the meaning of internal macros when referencing libraries (BSD and  
 96182 others)
- 96183 • Using a single instance of the shell for all of the command lines of the target (BSD and  
 96184 others)
- 96185 • Allowing <space> characters as well as <tab> characters to delimit command lines (BSD)
- 96186 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and  
 96187 others)
- 96188 • Remote execution of command lines (Sprite and others)
- 96189 • Specifying additional special targets (BSD, System V, and most others)

96190 Additionally, many vendors and research organizations have rethought the basic concepts of  
 96191 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of  
 96192 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of  
 96193 POSIX.1-2008 to require behavior that would be incompatible (and probably inferior) to  
 96194 historical practice for such a community.

96195 In similar circumstances, when the industry has enough sufficiently incompatible formats as to  
 96196 make them irreconcilable, this volume of POSIX.1-2008 has followed one or both of two courses  
 96197 of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options  
 96198 have been provided to select the desired behavior (*grep*, *od*, and *pax*).

96199 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes  
 96200 accepted by almost all versions of *make*, it was decided that it would be counter-productive to  
 96201 change the name. And since the makefile itself is a basic unit of portability, it would not be  
 96202 completely effective to reserve a new option letter, such as *make -P*, to achieve the portable  
 96203 behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to  
 96204 specify “standard” behavior. This special target does not preclude extensions in the *make* utility,  
 96205 nor does it preclude such extensions being used by the makefile specifying the target; it does,  
 96206 however, preclude any extensions from being applied that could alter the behavior of previously  
 96207 valid syntax; such extensions must be controlled via command line options or new special  
 96208 targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to  
 96209 guarantee that they are not affected by local extensions.

96210 The portable version of *make* described in this reference page is not intended to be the state-of-  
 96211 the-art software generation tool and, as such, some newer and more leading-edge features have  
 96212 not been included. An attempt has been made to describe the portable makefile in a manner that  
 96213 does not preclude such extensions as long as they do not disturb the portable behavior described

- 96214 here.
- 96215 When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive  
96216 `make -n target` to be used to see all of the action that would be taken to update `target`.
- 96217 The definition of `MAKEFLAGS` allows both the System V letter string and the BSD command  
96218 line formats. The two formats are sufficiently different to allow implementations to support both  
96219 without ambiguity.
- 96220 Early proposals stated that an “unquoted” `<number-sign>` was treated as the start of a  
96221 comment. The `make` utility does not pay any attention to quotes. A `<number-sign>` starts a  
96222 comment regardless of its surroundings.
- 96223 The text about “other implementation-defined pathnames may also be tried” in addition to  
96224 `./makefile` and `./Makefile` is to allow such extensions as `SCCS/s.Makefile` and other variations.  
96225 It was made an implementation-defined requirement (as opposed to unspecified behavior) to  
96226 highlight surprising implementations that might select something unexpected like  
96227 `/etc/Makefile`. XSI-conformant systems also try `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`,  
96228 and `SCCS/s.Makefile`.
- 96229 Early proposals contained the macro `NPROC` as a means of specifying that `make` should use `n`  
96230 processes to do the work required. While this feature is a valuable extension for many systems, it  
96231 is not common usage and could require other non-trivial extensions to makefile syntax. This  
96232 extension is not required by this volume of POSIX.1-2008, but could be provided as a compatible  
96233 extension. The macro `PARALLEL` is used by some historical systems with essentially the same  
96234 meaning (but without using a name that is a common system limit value). It is suggested that  
96235 implementors recognize the existing use of `NPROC` and/or `PARALLEL` as extensions to `make`.
- 96236 The default rules are based on System V. The default `CC=` value is `c99` instead of `cc` because this  
96237 volume of POSIX.1-2008 does not standardize the utility named `cc`. Thus, every conforming  
96238 application would be required to define `CC=c99` to expect to run. There is no advantage  
96239 conferred by the hope that the makefile might hit the “preferred” compiler because this cannot  
96240 be guaranteed to work. Also, since the portable makescript can only use the `c99` options, no  
96241 advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue  
96242 as to whether `c99` is as valuable as `cc`.
- 96243 The `-d` option to `make` is frequently used to produce debugging information, but is too  
96244 implementation-defined to add to this volume of POSIX.1-2008.
- 96245 The `-p` option is not passed in `MAKEFLAGS` on most historical implementations and to change  
96246 this would cause many implementations to break without sufficiently increased portability.
- 96247 Commands that begin with a `<plus-sign>` (`'+'`) are executed even if the `-n` option is present.  
96248 Based on the GNU version of `make`, the behavior of `-n` when the `<plus-sign>` prefix is  
96249 encountered has been extended to apply to `-q` and `-t` as well. However, the System V  
96250 convention of forcing command execution with `-n` when the command line of a target contains  
96251 either of the strings `"$(MAKE)"` or `"${MAKE}"` has not been adopted. This functionality  
96252 appeared in early proposals, but the danger of this approach was pointed out with the following  
96253 example of a portion of a makefile:
- ```
96254 subdir:
96255     cd subdir; rm all_the_files; $(MAKE)
```
- 96256 The loss of the System V behavior in this case is well-balanced by the safety afforded to other  
96257 makefiles that were not aware of this situation. In any event, the command line `<plus-sign>`  
96258 prefix can provide the desired functionality.
- 96259 The double `<colon>` in the target rule format is supported in BSD systems to allow more than

96260 one target line containing the same target name to have commands associated with it. Since this  
 96261 is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not  
 96262 mandated.

96263 The default rules are provided with text specifying that the built-in rules shall be the same as if  
 96264 the listed set were used. The intent is that implementations should be able to use the rules  
 96265 without change, but will be allowed to alter them in ways that do not affect the primary  
 96266 behavior.

96267 The best way to provide portable makefiles is to include all of the rules needed in the makefile  
 96268 itself. The rules provided use only features provided by other portions of this volume of  
 96269 POSIX.1-2008. The default rules include rules for optional commands in this volume of  
 96270 POSIX.1-2008. Only rules pertaining to commands that are provided are needed in the default  
 96271 set of an implementation.

96272 One point of discussion was whether to drop the default rules list from this volume of  
 96273 POSIX.1-2008. They provide convenience, but do not enhance portability of applications. The  
 96274 prime benefit is in portability of users who wish to type *make command* and have the command  
 96275 build from a **command.c** file.

96276 The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user  
 96277 override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the  
 96278 shell should be chosen by the makefile writer or specified on the *make* command line and not by  
 96279 a user running *make*.

96280 The *make* utilities in most historical implementations process the prerequisites of a target in left-  
 96281 to-right order, and the makefile format requires this. It supports the standard idiom used in  
 96282 many makefiles that produce *yacc* programs; for example:

```
96283 foo: y.tab.o lex.o main.o
96284      $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

96285 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct  
 96286 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used  
 96287 historically. Implementations that desire to update prerequisites in parallel should require an  
 96288 explicit extension to *make* or the makefile format to accomplish it, as described previously.

96289 The algorithm for determining a new entry for target rules is partially unspecified. Some  
 96290 historical *makes* allow blank, empty, or comment lines within the collection of commands  
 96291 marked by leading <tab> characters. A conforming makefile must ensure that each command  
 96292 starts with a <tab>, but implementations are free to ignore blank, empty, and comment lines  
 96293 without triggering the start of a new entry.

96294 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with  
 96295 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do  
 96296 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned  
 96297 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it  
 96298 is required to resend itself the signal it received so that it exits with a status that reflects the  
 96299 signal. The results from SIGQUIT are partially unspecified because, on systems that create **core**  
 96300 files upon receipt of SIGQUIT, the **core** from *make* would conflict with a **core** file from the  
 96301 command that was running when the SIGQUIT arrived. The main concern was to prevent  
 96302 damaged files from appearing up-to-date when *make* is rerun.

96303 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no  
 96304 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;  
 96305 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of  
 96306 targets than for the entire makefile. These extensions to *make* in System V were made to match

96307 historical practice from the BSD *make*.

96308 Macros are not exported to the environment of commands to be run. This was never the case in  
96309 any historical *make* and would have serious consequences. The environment is the same as the  
96310 environment to *make* except that *MAKEFLAGS* and macros defined on the *make* command line  
96311 are added.

96312 Some implementations do not use *system()* for all command lines, as required by the portable  
96313 makefile format; as a performance enhancement, they select lines without shell metacharacters  
96314 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but  
96315 merely that the same results be achieved. The metacharacters typically used to bypass the direct  
96316 *execve()* execution have been any of:

96317 = | ^ ( ) ; & < > \* ? [ ] : \$ \ ' " \ \n

96318 The default in some advanced versions of *make* is to group all the command lines for a target and  
96319 execute them using a single shell invocation; the System V method is to pass each line  
96320 individually to a separate shell. The single-shell method has the advantages in performance and  
96321 the lack of a requirement for many continued lines. However, converting to this newer method  
96322 has caused portability problems with many historical makefiles, so the behavior with the POSIX  
96323 makefile is specified to be the same as that of System V. It is suggested that the special target  
96324 *.ONESHELL* be used as an implementation extension to achieve the single-shell grouping for a  
96325 target or group of targets.

96326 Novice users of *make* have had difficulty with the historical need to start commands with a  
96327 <tab>. Since it is often difficult to discern differences between <tab> and <space> characters on  
96328 terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made  
96329 to correct this problem by allowing leading <blank> characters instead of <tab> characters.  
96330 However, implementors reported many makefiles that failed in subtle ways following this  
96331 change, and it is difficult to implement a *make* that unambiguously can differentiate between  
96332 macro and command lines. There is extensive historical practice of allowing leading <space>  
96333 characters before macro definitions. Forcing macro lines into column 1 would be a significant  
96334 backwards-compatibility problem for some makefiles. Therefore, historical practice was  
96335 restored.

96336 There is substantial variation in the handling of include lines by different implementations.  
96337 However, there is enough commonality for the standard to be able to specify a minimum set of  
96338 requirements that allow the feature to be used portably. Known variations have been explicitly  
96339 called out as unspecified behavior in the description.

96340 The System V dynamic dependency feature was not included. It would support:

96341 `cat: $$e.c`

96342 that would expand to;

96343 `cat: cat.c`

96344 This feature exists only in the new version of System V *make* and, while useful, is not in wide  
96345 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time  
96346 and once at target update time.

96347 Consideration was given to adding metarules to the POSIX *make*. This would make *%.o: %.c* the  
96348 same as *.c.o:*. This is quite useful and available from some vendors, but it would cause too many  
96349 changes to this *make* to support. It would have introduced rule chaining and new substitution  
96350 rules. However, the rules for target names have been set to reserve the '%' and '"' characters.  
96351 These are traditionally used to implement metarules and quoting of target names, respectively.  
96352 Implementors are strongly encouraged to use these characters only for these purposes.

- 96353 A request was made to extend the suffix delimiter character from a <period> to any character.  
96354 The metarules feature in newer *makes* solves this problem in a more general way. This volume of  
96355 POSIX.1-2008 is staying with the more conservative historical definition.
- 96356 The standard output format for the `-p` option is not described because it is primarily a  
96357 debugging option and because the format is not generally useful to programs. In historical  
96358 implementations the output is not suitable for use in generating makefiles. The `-p` format has  
96359 been variable across historical implementations. Therefore, the definition of `-p` was only to  
96360 provide a consistently named option for obtaining *make* script debugging information.
- 96361 Some historical implementations have not cleared the suffix list with `-r`.
- 96362 Implementations should be aware that some historical applications have intermixed *target\_name*  
96363 and *macro=value* operands on the command line, expecting that all of the macros are processed  
96364 before any of the targets are dealt with. Conforming applications do not do this, but some  
96365 backwards-compatibility support may be warranted.
- 96366 Empty inference rules are specified with a <semicolon> command rather than omitting all  
96367 commands, as described in an early proposal. The latter case has no traditional meaning and is  
96368 reserved for implementation extensions, such as in GNU *make*.
- 96369 **FUTURE DIRECTIONS**
- 96370 None.
- 96371 **SEE ALSO**
- 96372 [Chapter 2](#) (on page 2297), *ar*, *c99*, *get*, *lex*, *sccs*, *sh*, *yacc*
- 96373 [XBD Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 96374 [XSH \*exec\*, \*system\(\)\*](#)
- 96375 **CHANGE HISTORY**
- 96376 First released in Issue 2.
- 96377 **Issue 5**
- 96378 The FUTURE DIRECTIONS section is added.
- 96379 **Issue 6**
- 96380 This utility is marked as part of the Software Development Utilities option.
- 96381 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the  
96382 SPECIAL TARGETS section.
- 96383 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
96384 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of  
96385 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.
- 96386 It is specified whether the command line is related to the makefile or to the *make* command, and  
96387 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 96388 The normative text is reworded to avoid use of the term “must” for application requirements.
- 96389 PASC Interpretation 1003.2 #193 is applied.
- 96390 **Issue 7**
- 96391 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
96392 apply.
- 96393 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 96394 Include lines in makefiles are introduced.

96395  
96396

Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution** section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

96397 **NAME**

96398 man — display system documentation

96399 **SYNOPSIS**96400 man [-k] *name* . . .96401 **DESCRIPTION**

96402 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a  
 96403 standard utility, *man* at a minimum shall write a message describing the syntax used by the  
 96404 standard utility, its options, and operands. If more information is available, the *man* utility shall  
 96405 provide it in an implementation-defined manner.

96406 An implementation may provide information for values of *name* other than the standard utilities.  
 96407 Standard utilities that are listed as optional and that are not supported by the implementation  
 96408 either shall cause a brief message indicating that fact to be displayed or shall cause a full display  
 96409 of information as described previously.

96410 **OPTIONS**96411 The *man* utility shall conform to XBD Section 12.2 (on page 215).

96412 The following option shall be supported:

96413 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary  
 96414 database that contains a brief purpose entry for each standard utility and write lines  
 96415 from the summary database that match any of the keywords. The keyword search shall  
 96416 produce results that are the equivalent of the output of the following command:

```
96417 grep -Ei '  
96418     name  
96419     name  
96420     . . .  
96421     ' summary-database
```

96422 This assumes that the *summary-database* is a text file with a single entry per line; this  
 96423 organization is not required and the example using *grep -Ei* is merely illustrative of the  
 96424 type of search intended. The purpose entry to be included in the database shall consist  
 96425 of a terse description of the purpose of the utility.

96426 **OPERANDS**

96427 The following operand shall be supported:

96428 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*  
 96429 does not represent one of the standard utilities, the results are unspecified.

96430 **STDIN**

96431 Not used.

96432 **INPUT FILES**

96433 None.

96434 **ENVIRONMENT VARIABLES**96435 The following environment variables shall affect the execution of *man*:

96436 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 96437 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 96438 variables used to determine the values of locale categories.)

96439 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 96440 internationalization variables.

- 96441 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 96442 characters (for example, single-byte as opposed to multi-byte characters in  
 96443 arguments and in the summary database). The value of *LC\_CTYPE* need not affect  
 96444 the format of the information written about the *name* operands.
- 96445 **LC\_MESSAGES**  
 96446 Determine the locale that should be used to affect the format and contents of  
 96447 diagnostic messages written to standard error and informative messages written to  
 96448 standard output.
- 96449 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 96450 **PAGER** Determine an output filtering command for writing the output to a terminal. Any  
 96451 string acceptable as a *command\_string* operand to the *sh -c* command shall be valid.  
 96452 When standard output is a terminal device, the reference page output shall be  
 96453 piped through the command. If the *PAGER* variable is null or not set, the command  
 96454 shall be either *more* or another paginator utility documented in the system  
 96455 documentation.
- 96456 **ASYNCHRONOUS EVENTS**  
 96457 Default.
- 96458 **STDOUT**  
 96459 The *man* utility shall write text describing the syntax of the utility *name*, its options and its  
 96460 operands, or, when *-k* is specified, lines from the summary database. The format of this text is  
 96461 implementation-defined.
- 96462 **STDERR**  
 96463 The standard error shall be used for diagnostic messages, and may also be used for  
 96464 informational messages of unspecified format.
- 96465 **OUTPUT FILES**  
 96466 None.
- 96467 **EXTENDED DESCRIPTION**  
 96468 None.
- 96469 **EXIT STATUS**  
 96470 The following exit values shall be returned:  
 96471 0 Successful completion.  
 96472 >0 An error occurred.
- 96473 **CONSEQUENCES OF ERRORS**  
 96474 Default.
- 96475 **APPLICATION USAGE**  
 96476 None.
- 96477 **EXAMPLES**  
 96478 None.
- 96479 **RATIONALE**  
 96480 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the  
 96481 standard developers was strongly divided as to how much or how little information *man* should  
 96482 be required to provide. They considered, however, that the provision of some portable way of  
 96483 accessing documentation would aid user portability. The arguments against a fuller specification  
 96484 were:

- 96485 • Large quantities of documentation should not be required on a system that does not have
- 96486 excess disk space.
- 96487 • The current manual system does not present information in a manner that greatly aids user
- 96488 portability.
- 96489 • A “better help system” is currently an area in which vendors feel that they can add value
- 96490 to their POSIX implementations.

96491 The `-f` option was considered, but due to implementation differences, it was not included in this  
 96492 volume of POSIX.1-2008.

96493 The description was changed to be more specific about what has to be displayed for a utility. The  
 96494 standard developers considered it insufficient to allow a display of only the synopsis without  
 96495 giving a short description of what each option and operand does.

96496 The “purpose” entry to be included in the database can be similar to the section title (less the  
 96497 numeric prefix) from this volume of POSIX.1-2008 for each utility. These titles are similar to  
 96498 those used in historical systems for this purpose.

96499 See *mailx* for rationale concerning the default paginator.

96500 The caveat in the *LC\_CTYPE* description was added because it is not a requirement that an  
 96501 implementation provide reference pages for all of its supported locales on each system;  
 96502 changing *LC\_CTYPE* does not necessarily translate the reference page into another language.  
 96503 This is equivalent to the current state of *LC\_MESSAGES* in POSIX.1-2008—locale-specific  
 96504 messages are not yet a requirement.

96505 The historical *MANPATH* variable is not included in POSIX because no attempt is made to  
 96506 specify naming conventions for reference page files, nor even to mandate that they are files at  
 96507 all. On some implementations they could be a true database, a hypertext file, or even fixed  
 96508 strings within the *man* executable. The standard developers considered the portability of  
 96509 reference pages to be outside their scope of work. However, users should be aware that  
 96510 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor  
 96511 the search pattern for reference pages from the various categories (utilities, functions, file  
 96512 formats, and so on) when the system administrator reveals the location and conventions for  
 96513 reference pages on the system.

96514 The keyword search can rely on at least the text of the section titles from these utility  
 96515 descriptions, and the implementation may add more keywords. The term “section titles” refers  
 96516 to the strings such as:

```
96517 man - Display system documentation
96518 ps - Report process status
```

#### 96519 FUTURE DIRECTIONS

96520 None.

#### 96521 SEE ALSO

96522 *more*

96523 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

#### 96524 CHANGE HISTORY

96525 First released in Issue 4.

96526 **Issue 5**

96527 The FUTURE DIRECTIONS section is added.

96528 **Issue 7**96529 Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages  
96530 may appear on standard error.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**mesg**96531 **NAME**

96532           mesg — permit or deny messages

96533 **SYNOPSIS**

96534           mesg [y|n]

96535 **DESCRIPTION**

96536           The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or  
 96537           other utilities to a terminal device. The terminal device affected shall be determined by searching  
 96538           for the first terminal in the sequence of devices associated with standard input, standard output,  
 96539           and standard error, respectively. With no arguments, *mesg* shall report the current state without  
 96540           changing it. Processes with appropriate privileges may be able to send messages to the terminal  
 96541           independent of the current state.

96542 **OPTIONS**

96543           None.

96544 **OPERANDS**

96545           The following operands shall be supported in the POSIX locale:

96546           *y*           Grant permission to other users to send messages to the terminal device.96547           *n*           Deny permission to other users to send messages to the terminal device.96548 **STDIN**

96549           Not used.

96550 **INPUT FILES**

96551           None.

96552 **ENVIRONMENT VARIABLES**96553           The following environment variables shall affect the execution of *mesg*:

96554           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 96555                       (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 96556                       variables used to determine the values of locale categories.)

96557           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 96558                       internationalization variables.

96559           *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 96560                       characters (for example, single-byte as opposed to multi-byte characters in  
 96561                       arguments).

96562           *LC\_MESSAGES*

96563                       Determine the locale that should be used to affect the format and contents of  
 96564                       diagnostic messages written (by *mesg*) to standard error.

96565           XSI       *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

96566 **ASYNCHRONOUS EVENTS**

96567           Default.

96568 **STDOUT**96569           If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.96570 **STDERR**

96571           The standard error shall be used only for diagnostic messages.

96572 **OUTPUT FILES**

96573 None.

96574 **EXTENDED DESCRIPTION**

96575 None.

96576 **EXIT STATUS**

96577 The following exit values shall be returned:

96578 0 Receiving messages is allowed.

96579 1 Receiving messages is not allowed.

96580 &gt;1 An error occurred.

96581 **CONSEQUENCES OF ERRORS**

96582 Default.

96583 **APPLICATION USAGE**

96584 The mechanism by which the message status of the terminal is changed is unspecified.  
 96585 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has  
 96586 successfully completed. These actions may include, but are not limited to: another invocation of  
 96587 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or  
 96588 *chmod()* function, and so on.

96589 **EXAMPLES**

96590 None.

96591 **RATIONALE**

96592 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather  
 96593 than the controlling terminal for the session. This is because users logged in more than once  
 96594 should be able to change any of their login terminals without having to stop the job running in  
 96595 those sessions. This is not a security problem involving the terminals of other users because  
 96596 appropriate privileges would be required to affect the terminal of another user.

96597 The method of checking each of the first three file descriptors in sequence until a terminal is  
 96598 found was adopted from System V.

96599 The file */dev/tty* is not specified for the terminal device because it was thought to be too  
 96600 restrictive. Typical environment changes for the *n* operand are that write permissions are  
 96601 removed for *others* and *group* from the appropriate device. It was decided to leave the actual  
 96602 description of what is done as unspecified because of potential differences between  
 96603 implementations.

96604 The format for standard output is unspecified because of differences between historical  
 96605 implementations. This output is generally not useful to shell scripts (they can use the exit status),  
 96606 so exact parsing of the output is unnecessary.

96607 **FUTURE DIRECTIONS**

96608 None.

96609 **SEE ALSO**96610 *talk*, *write*

96611 XBD Chapter 8 (on page 173)

**mesg***Utilities*96612 **CHANGE HISTORY**

96613 First released in Issue 2.

96614 **Issue 6**

96615 This utility is marked as part of the User Portability Utilities option.

96616 **Issue 7**96617 The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability  
96618 Utilities is now an option for interactive utilities.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

96619 **NAME**

96620 mkdir — make directories

96621 **SYNOPSIS**96622 mkdir [-p] [-m *mode*] *dir*...96623 **DESCRIPTION**96624 The *mkdir* utility shall create the directories specified by the operands, in the order specified.96625 For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function  
96626 defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

- 96627 1. The *dir* operand is used as the *path* argument.
- 96628 2. The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO is used as  
96629 the *mode* argument. (If the **-m** option is specified, the value of the *mkdir()* *mode* argument  
96630 is unspecified, but the directory shall at no time have permissions less restrictive than the  
96631 **-m mode** option-argument.)

96632 **OPTIONS**96633 The *mkdir* utility shall conform to XBD Section 12.2 (on page 215).

96634 The following options shall be supported:

96635 **-m mode** Set the file permission bits of the newly-created directory to the specified *mode*  
96636 value. The *mode* option-argument shall be the same as the *mode* operand defined  
96637 for the *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-'  
96638 shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add  
96639 permissions to the default mode; '-' shall delete permissions from the default  
96640 mode.

96641 **-p** Create any missing intermediate pathname components.

96642 For each *dir* operand that does not name an existing directory, effects equivalent to  
96643 those caused by the following command shall occur:

```
96644 mkdir -p -m $(umask -S),u+wx $(dirname dir) &&
96645 mkdir [-m mode] dir
```

96646 where the **-m mode** option represents that option supplied to the original  
96647 invocation of *mkdir*, if any.

96648 Each *dir* operand that names an existing directory shall be ignored without error.

96649 **OPERANDS**

96650 The following operand shall be supported:

96651 *dir* A pathname of a directory to be created.

96652 **STDIN**

96653 Not used.

96654 **INPUT FILES**

96655 None.

96656 **ENVIRONMENT VARIABLES**96657 The following environment variables shall affect the execution of *mkdir*:

96658 **LANG** Provide a default value for the internationalization variables that are unset or null.  
96659 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
96660 variables used to determine the values of locale categories.)

**mkdir**

Utilities

96661	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
96662		
96663	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
96664		
96665		
96666	<i>LC_MESSAGES</i>	
96667		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
96668		
96669	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
96670	<b>ASYNCHRONOUS EVENTS</b>	
96671		Default.
96672	<b>STDOUT</b>	
96673		Not used.
96674	<b>STDERR</b>	
96675		The standard error shall be used only for diagnostic messages.
96676	<b>OUTPUT FILES</b>	
96677		None.
96678	<b>EXTENDED DESCRIPTION</b>	
96679		None.
96680	<b>EXIT STATUS</b>	
96681		The following exit values shall be returned:
96682	0	All the specified directories were created successfully or the <b>-p</b> option was specified and all the specified directories now exist.
96683		
96684	>0	An error occurred.
96685	<b>CONSEQUENCES OF ERRORS</b>	
96686		Default.
96687	<b>APPLICATION USAGE</b>	
96688		The default file mode for directories is <i>a=rwx</i> (777 on most systems) with selected permissions removed in accordance with the file mode creation mask. For intermediate pathname components created by <i>mkdir</i> , the mode is the default modified by <i>u+wx</i> so that the subdirectories can always be created regardless of the file mode creation mask; if different ultimate permissions are desired for the intermediate directories, they can be changed afterwards with <i>chmod</i> .
96689		
96690		
96691		
96692		
96693		
96694		Note that some of the requested directories may have been created even if an error occurs.
96695	<b>EXAMPLES</b>	
96696		None.
96697	<b>RATIONALE</b>	
96698		The System V <b>-m</b> option was included to control the file mode.
96699		The System V <b>-p</b> option was included to create any needed intermediate directories and to complement the functionality provided by <i>rmdir</i> for removing directories in the path prefix as they become empty. Because no error is produced if any path component already exists, the <b>-p</b> option is also useful to ensure that a particular directory exists.
96700		
96701		
96702		
96703		The functionality of <i>mkdir</i> is described substantially through a reference to the <i>mkdir()</i> function

96704 in the System Interfaces volume of POSIX.1-2008. For example, by default, the mode of the  
96705 directory is affected by the file mode creation mask in accordance with the specified behavior of  
96706 the *mkdir()* function. In this way, there is less duplication of effort required for describing details  
96707 of the directory creation.

96708 **FUTURE DIRECTIONS**

96709 None.

96710 **SEE ALSO**

96711 *chmod, rm, rmdir, umask*

96712 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

96713 XSH *mkdir()*

96714 **CHANGE HISTORY**

96715 First released in Issue 2.

96716 **Issue 5**

96717 The FUTURE DIRECTIONS section is added.

96718 **Issue 7**

96719 SD5-XCU-ERN-56 is applied, aligning the *-m* option with the IEEE P1003.2b draft standard to  
96720 clarify an ambiguity.

96721 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**mkfifo**

Utilities

96722 **NAME**

96723 mkfifo — make FIFO special files

96724 **SYNOPSIS**96725 mkfifo [-m *mode*] *file*...96726 **DESCRIPTION**96727 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order  
96728 specified.96729 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function  
96730 defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

- 96731 1. The *file* operand is used as the *path* argument.
- 96732 2. The value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
96733 S\_IROTH, and S\_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the  
96734 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have  
96735 permissions less restrictive than the **-m mode** option-argument.)

96736 **OPTIONS**96737 The *mkfifo* utility shall conform to XBD Section 12.2 (on page 215).

96738 The following option shall be supported:

96739 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.  
96740 The *mode* option-argument shall be the same as the *mode* operand defined for the  
96741 *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-' shall be  
96742 interpreted relative to an assumed initial mode of *a=rw*.

96743 **OPERANDS**

96744 The following operand shall be supported:

96745 *file* A pathname of the FIFO special file to be created.96746 **STDIN**

96747 Not used.

96748 **INPUT FILES**

96749 None.

96750 **ENVIRONMENT VARIABLES**96751 The following environment variables shall affect the execution of *mkfifo*:

96752 **LANG** Provide a default value for the internationalization variables that are unset or null.  
96753 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
96754 variables used to determine the values of locale categories.)

96755 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
96756 internationalization variables.

96757 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
96758 characters (for example, single-byte as opposed to multi-byte characters in  
96759 arguments).

96760 **LC\_MESSAGES**

96761 Determine the locale that should be used to affect the format and contents of  
96762 diagnostic messages written to standard error.

- 96763 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 96764 **ASYNCHRONOUS EVENTS**
- 96765 Default.
- 96766 **STDOUT**
- 96767 Not used.
- 96768 **STDERR**
- 96769 The standard error shall be used only for diagnostic messages.
- 96770 **OUTPUT FILES**
- 96771 None.
- 96772 **EXTENDED DESCRIPTION**
- 96773 None.
- 96774 **EXIT STATUS**
- 96775 The following exit values shall be returned:
- 96776 0 All the specified FIFO special files were created successfully.
- 96777 >0 An error occurred.
- 96778 **CONSEQUENCES OF ERRORS**
- 96779 Default.
- 96780 **APPLICATION USAGE**
- 96781 None.
- 96782 **EXAMPLES**
- 96783 None.
- 96784 **RATIONALE**
- 96785 This utility was added to permit shell applications to create FIFO special files.
- 96786 The **-m** option was added to control the file mode, for consistency with the similar functionality provided by the *mkdir* utility.
- 96787
- 96788 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.
- 96789
- 96790
- 96791 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of POSIX.1-2008. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.
- 96792
- 96793
- 96794
- 96795
- 96796 **FUTURE DIRECTIONS**
- 96797 None.
- 96798 **SEE ALSO**
- 96799 *chmod*, *umask*
- 96800 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
- 96801 XSH *mkfifo()*

96802 **CHANGE HISTORY**

96803 First released in Issue 3.

96804 **Issue 6**

96805 The **-m** option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

96806 **NAME**96807 `more` — display files on a page-by-page basis96808 **SYNOPSIS**96809 UP `more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]`96810 **DESCRIPTION**

96811 The *more* utility shall read files and either write them to the terminal on a page-by-page basis or  
 96812 filter them to standard output. If standard output is not a terminal device, all input files shall be  
 96813 copied to standard output in their entirety, without modification, except as specified for the `-s`  
 96814 option. If standard output is a terminal device, the files shall be written a number of lines (one  
 96815 screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION  
 96816 section.

96817 Certain block-mode terminals do not have all the capabilities necessary to support the complete  
 96818 *more* definition; they are incapable of accepting commands that are not terminated with a  
 96819 `<newline>`. Implementations that support such terminals shall provide an operating mode to  
 96820 *more* in which all commands can be terminated with a `<newline>` on those terminals. This mode:

- 96821 • Shall be documented in the system documentation
- 96822 • Shall, at invocation, inform the user of the terminal deficiency that requires the `<newline>`  
 96823 usage and provide instructions on how this warning can be suppressed in future  
 96824 invocations
- 96825 • Shall not be required for implementations supporting only fully capable terminals
- 96826 • Shall not affect commands already requiring `<newline>` characters
- 96827 • Shall not affect users on the capable terminals from using *more* as described in this volume  
 96828 of POSIX.1-2008

96829 **OPTIONS**

96830 The *more* utility shall conform to XBD [Section 12.2](#) (on page 215), except that `'+'` may be  
 96831 recognized as an option delimiter as well as `'-'`.

96832 The following options shall be supported:

- 96833 `-c` If a screen is to be written that has no lines in common with the current screen, or  
 96834 *more* is writing its first screen, *more* shall not scroll the screen, but instead shall  
 96835 redraw each line of the screen in turn, from the top of the screen to the bottom. In  
 96836 addition, if *more* is writing its first screen, the screen shall be cleared. This option  
 96837 may be silently ignored on devices with insufficient terminal capabilities.
- 96838 `-e` By default, *more* shall exit immediately after writing the last line of the last file in  
 96839 the argument list. If the `-e` option is specified:
  - 96840 1. If there is only a single file in the argument list and that file was completely  
 96841 displayed on a single screen, *more* shall exit immediately after writing the  
 96842 last line of that file.
  - 96843 2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in  
 96844 the argument list twice without an intervening operation. See the  
 96845 EXTENDED DESCRIPTION section.
- 96846 `-i` Perform pattern matching in searches without regard to case; see XBD [Section 9.2](#)  
 96847 (on page 182).



96892 **INPUT FILES**

96893 The input files being examined shall be text files. If standard output is a terminal, standard error  
 96894 shall be used to read commands from the user. If standard output is a terminal, standard error is  
 96895 not readable, and command input is needed, *more* may attempt to obtain user commands from  
 96896 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error  
 96897 indicating that it was unable to read user commands. If standard output is not a terminal, no  
 96898 error shall result if standard error cannot be opened for reading.

96899 **ENVIRONMENT VARIABLES**

96900 The following environment variables shall affect the execution of *more*:

96901 *COLUMNS* Override the system-selected horizontal display line size. See XBD Chapter 8 (on  
 96902 page 173) for valid values and results when it is unset or null.

96903 *EDITOR* Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION  
 96904 section.

96905 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 96906 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 96907 variables used to determine the values of locale categories.)

96908 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 96909 internationalization variables.

96910 *LC\_COLLATE*

96911 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 96912 character collating elements within regular expressions.

96913 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 96914 characters (for example, single-byte as opposed to multi-byte characters in  
 96915 arguments and input files) and the behavior of character classes within regular  
 96916 expressions.

96917 *LC\_MESSAGES*

96918 Determine the locale that should be used to affect the format and contents of  
 96919 diagnostic messages written to standard error and informative messages written to  
 96920 standard output.

96921 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

96922 *LINES* Override the system-selected vertical screen size, used as the number of lines in a  
 96923 screenful. See XBD Chapter 8 (on page 173) for valid values and results when it is  
 96924 unset or null. The *-n* option shall take precedence over the *LINES* variable for  
 96925 determining the number of lines in a screenful.

96926 *MORE*

96927 Determine a string containing options described in the OPTIONS section preceded  
 96928 with <hyphen> characters and <blank>-separated as on the command line. Any  
 96929 command line options shall be processed after those in the *MORE* variable, as if  
 the command line were:

96930 `more $MORE options operands`

96931 The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for  
 96932 determining the number of lines in a screenful.

96933 *TERM* Determine the name of the terminal type. If this variable is unset or null, an  
 96934 unspecified default terminal type is used.

96935 **ASYNCHRONOUS EVENTS**

96936 Default.

96937 **STDOUT**

96938 The standard output shall be used to write the contents of the input files.

96939 **STDERR**

96940 The standard error shall be used for diagnostic messages and user commands (see the INPUT  
 96941 FILES section), and, if standard output is a terminal device, to write a prompting string. The  
 96942 prompting string shall appear on the screen line below the last line of the file displayed in the  
 96943 current screenful. The prompt shall contain the name of the file currently being examined and  
 96944 shall contain an end-of-file indication and the name of the next file, if any, when prompting at  
 96945 the end-of-file. If an error or informational message is displayed, it is unspecified whether it is  
 96946 contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the  
 96947 user shall be prompted for a continuation character, at which point another message or the user  
 96948 prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether  
 96949 informational messages are written for other user commands.

96950 **OUTPUT FILES**

96951 None.

96952 **EXTENDED DESCRIPTION**

96953 The following section describes the behavior of *more* when the standard output is a terminal  
 96954 device. If the standard output is not a terminal device, no options other than `-s` shall have any  
 96955 effect, and all input files shall be copied to standard output otherwise unmodified, at which time  
 96956 *more* shall exit without further action.

96957 The number of lines available per screen shall be determined by the `-n` option, if present, or by  
 96958 examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither  
 96959 method yields a number, an unspecified number of lines shall be used.

96960 The maximum number of lines written shall be one less than this number, because the screen  
 96961 line after the last line written shall be used to write a user prompt and user input. If the number  
 96962 of lines in the screen is less than two, the results are undefined. It is unspecified whether user  
 96963 input is permitted to be longer than the remainder of the single line where the prompt has been  
 96964 written.

96965 The number of columns available per line shall be determined by examining values in the  
 96966 environment (see the ENVIRONMENT VARIABLES section), with a default value as described  
 96967 in XBD Chapter 8 (on page 173).

96968 Lines that are longer than the display shall be folded; the length at which folding occurs is  
 96969 unspecified, but should be appropriate for the output device. Folding may occur between glyphs  
 96970 of single characters that take up multiple display columns.

96971 When standard output is a terminal and `-u` is not specified, *more* shall treat `<backspace>` and  
 96972 `<carriage-return>` characters specially:

- 96973 • A character, followed first by a sequence of  $n$  `<backspace>` characters (where  $n$  is the same  
 96974 as the number of column positions that the character occupies), then by  $n$  `<underscore>`  
 96975 characters (`'_'`), shall cause that character to be written as underlined text, if the terminal  
 96976 type supports that. The  $n$  `<underscore>` characters, followed first by  $n$  `<backspace>`  
 96977 characters, then any character with  $n$  column positions, shall also cause that character to be  
 96978 written as underlined text, if the terminal type supports that.

- 96979 • A sequence of  $n$  <backspace> characters (where  $n$  is the same as the number of column  
96980 positions that the previous character occupies) that appears between two identical  
96981 printable characters shall cause the first of those two characters to be written as  
96982 emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the  
96983 terminal type supports that, and the second to be discarded. Immediately subsequent  
96984 occurrences of <backspace>/character pairs for that same character shall also be  
96985 discarded. (For example, the sequence "a\ba\ba\ba" is interpreted as a single  
96986 emboldened 'a'.)
- 96987 • The *more* utility shall logically discard all other <backspace> characters from the line as  
96988 well as the character which precedes them, if any.
- 96989 • A <carriage-return> at the end of a line shall be ignored, rather than being written as a  
96990 non-printable character, as described in the next paragraph.

96991 It is implementation-defined how other non-printable characters are written. Implementations  
96992 should use the same format that they use for the *ex print* command; see the OPTIONS section  
96993 within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it  
96994 crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the  
96995 number of columns on the display is less than the number of columns any single character in the  
96996 line being displayed would occupy.

96997 When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.  
96998 Once the initial screen has been written, *more* shall prompt for a user command. If the execution  
96999 of the user command results in a screen that has lines in common with the current screen, and  
97000 the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is  
97001 unspecified whether the screen is scrolled or redrawn.

97002 For all files but the last (including standard input if no file was specified, and for the last file as  
97003 well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall  
97004 prompt for a user command. This prompt shall contain the name of the next file as well as an  
97005 indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*,  
97006 <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last  
97007 file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

97008 Several of the commands described in this section display a previous screen from the input  
97009 stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is  
97010 implementation-defined how much backwards motion is supported. If a command cannot be  
97011 executed because of a limitation on backwards motion, an error message to this effect shall be  
97012 displayed, the current screen shall not change, and the user shall be prompted for another  
97013 command.

97014 If a command cannot be performed because there are insufficient lines to display, *more* shall alert  
97015 the terminal. If a command cannot be performed because there are insufficient lines to display or  
97016 a / command fails: if the input is the standard input, the last screen in the file may be displayed;  
97017 otherwise, the current file and screen shall not change, and the user shall be prompted for  
97018 another command.

97019 The interactive commands in the following sections shall be supported. Some commands can be  
97020 preceded by a decimal integer, called *count* in the following descriptions. If not specified with  
97021 the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular  
97022 expression, as described in XBD Section 9.3 (on page 183). The term "examine" is historical  
97023 usage meaning "open the file for viewing"; for example, *more foo* would be expressed as  
97024 examining file *foo*.

97025 In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a

97026 line from the file being examined.

97027 In the following descriptions, the *current position* refers to two things:

- 97028 1. The position of the current line on the screen
- 97029 2. The line number (in the file) of the current line on the screen

97030 Usually, the line on the screen corresponding to the current position is the third line on the  
 97031 screen. If this is not possible (there are fewer than three lines to display or this is the first page of  
 97032 the file, or it is the last page of the file), then the current position is either the first or last line on  
 97033 the screen as described later.

### 97034 Help

97035 *Synopsis:*     h

97036 Write a summary of these commands and other implementation-defined commands. The  
 97037 behavior shall be as if the *more* utility were executed with the *-e* option on a file that contained  
 97038 the summary information. The user shall be prompted as described earlier in this section when  
 97039 end-of-file is reached. If the user command is one of those specified to continue to the next file,  
 97040 *more* shall return to the file and screen state from which the *h* command was executed.

### 97041 Scroll Forward One Screenful

97042 *Synopsis:*     [*count*] f  
 97043                 [*count*] <control>--F

97044 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,  
 97045 only the final screenful shall be written.

### 97046 Scroll Backward One Screenful

97047 *Synopsis:*     [*count*] b  
 97048                 [*count*] <control>--B

97049 Scroll backward *count* lines, with a default of one screenful (see the *-n* option). If *count* is more  
 97050 than the screen size, only the final screenful shall be written.

### 97051 Scroll Forward One Line

97052 *Synopsis:*     [*count*] <space>  
 97053                 [*count*] j  
 97054                 [*count*] <newline>

97055 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for *j* and  
 97056 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen  
 97057 size.

### 97058 Scroll Backward One Line

97059 *Synopsis:*     [*count*] k

97060 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the  
 97061 screen size.

97062 **Scroll Forward One Half Screenful**

97063 *Synopsis:*     [*count*]d  
 97064                   [*count*]<control>-D

97065 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 97066 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

97067 **Skip Forward One Line**

97068 *Synopsis:*     [*count*]s

97069 Display the screenful beginning with the line *count* lines after the last line on the current screen.  
 97070 If *count* would cause the current position to be such that less than one screenful would be  
 97071 written, the last screenful in the file shall be written.

97072 **Scroll Backward One Half Screenful**

97073 *Synopsis:*     [*count*]u  
 97074                   [*count*]<control>-U

97075 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 97076 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.  
 97077 The entire *count* lines shall be written, even if *count* is more than the screen size.

97078 **Go to Beginning of File**

97079 *Synopsis:*     [*count*]g

97080 Display the screenful beginning with line *count*.

97081 **Go to End-of-File**

97082 *Synopsis:*     [*count*]G

97083 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the  
 97084 last screenful of the file.

97085 **Refresh the Screen**

97086 *Synopsis:*     r  
 97087                   <control>-L

97088 Refresh the screen.

97089 **Discard and Refresh**

97090 *Synopsis:*     R

97091 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered  
 97092 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

97093 **Mark Position**97094 *Synopsis:* `mletter`

97095 Mark the current position with the letter named by *letter*, where *letter* represents the name of one  
 97096 of the lowercase letters of the portable character set. When a new file is examined, all marks may  
 97097 be lost.

97098 **Return to Mark**97099 *Synopsis:* `' letter`

97100 Return to the position that was previously marked with the letter named by *letter*, making that  
 97101 line the current position.

97102 **Return to Previous Position**97103 *Synopsis:* `''`

97104 Return to the position from which the last large movement command was executed (where a  
 97105 "large movement" is defined as any movement of more than a screenful of lines). If no such  
 97106 movements have been made, return to the beginning of the file.

97107 **Search Forward for Pattern**97108 *Synopsis:* `[count]/[!]pattern<newline>`

97109 Display the screenful beginning with the *count*th line containing the pattern. The search shall  
 97110 start after the first line currently displayed. The null regular expression (`'/'` followed by a  
 97111 `<newline>`) shall repeat the search using the previous regular expression, with a default *count*. If  
 97112 the character `'!'` is included, the matching lines shall be those that do not contain the *pattern*. If  
 97113 no match is found for the *pattern*, a message to that effect shall be displayed.

97114 **Search Backward for Pattern**97115 *Synopsis:* `[count]?[!]pattern<newline>`

97116 Display the screenful beginning with the *count*th previous line containing the pattern. The search  
 97117 shall start on the last line before the first line currently displayed. The null regular expression  
 97118 (`'?'` followed by a `<newline>`) shall repeat the search using the previous regular expression,  
 97119 with a default *count*. If the character `'!'` is included, matching lines shall be those that do not  
 97120 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be  
 97121 displayed.

97122 **Repeat Search**97123 *Synopsis:* `[count]n`

97124 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last  
 97125 *pattern*, if the previous search was `"!/"` or `"?!"`).

97126 **Repeat Search in Reverse**97127 *Synopsis:*     [*count*]N97128 Repeat the search in the opposite direction of the previous search for the *count*th line containing  
97129 the last *pattern* (or not containing the last *pattern*, if the previous search was `"/!"` or `"?!"`).97130 **Examine New File**97131 *Synopsis:*     :e [*filename*]  
<newline>97132 Examine a new file. If the *filename* argument is not specified, the current file (see the `:n` and `:p`  
97133 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell  
97134 word expansions (see Section 2.6, on page 2305); if more than a single pathname results, the  
97135 effects are unspecified. If *filename* is a <number-sign> (`'#'`), the previously examined file shall  
97136 be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable  
97137 file), an error message to this effect shall be displayed and the current file and screen shall not  
97138 change.97139 **Examine Next File**97140 *Synopsis:*     [*count*]:n97141 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If  
97142 *filename* refers to a non-seekable file, the results are unspecified.97143 **Examine Previous File**97144 *Synopsis:*     [*count*]:p97145 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be  
97146 examined. If *filename* refers to a non-seekable file, the results are unspecified.97147 **Go to Tag**97148 *Synopsis:*     :t *tagstring*  
<newline>97149 If the file containing the tag named by the *tagstring* argument is not the current file, examine the  
97150 file, as if the `:e` command was executed with that file as the argument. Otherwise, or in addition,  
97151 display the screenful beginning with the tag, as described for the `-t` option (see the OPTIONS  
97152 section). If the *ctags* utility is not supported by the system, the use of `:t` produces undefined  
97153 results.97154 **Invoke Editor**97155 *Synopsis:*     v97156 Invoke an editor to edit the current file being examined. If standard input is being examined, the  
97157 results are unspecified. The name of the editor shall be taken from the environment variable  
97158 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the  
97159 editor shall be invoked with a `-c linenum` command line argument, where *linenum* is the  
97160 line number of the file line containing the display line currently displayed as the first line of the  
97161 screen. It is implementation-defined whether line-setting options are passed to editors other  
97162 than *vi* and *ex*.97163 When the editor exits, *more* shall resume with the same file and screen as when the editor was  
97164 invoked.

97165 **Display Position**

97166 *Synopsis:* =  
 97167 <control>-G

97168 Write a message for which the information references the first byte of the line after the last line of  
 97169 the file on the screen. This message shall include the name of the file currently being examined,  
 97170 its number relative to the total number of files there are to examine, the line number in the file  
 97171 the byte number and the total bytes in the file, and what percentage of the file precedes the  
 97172 current position. If *more* is reading from standard input, or the file is shorter than a single screen,  
 97173 the line number, the byte number, the total bytes, and the percentage need not be written.

97174 **Quit**

97175 *Synopsis:* q  
 97176 :q  
 97177 ZZ

97178 Exit *more*.

97179 **EXIT STATUS**

97180 The following exit values shall be returned:

97181 0 Successful completion.

97182 >0 An error occurred.

97183 **CONSEQUENCES OF ERRORS**

97184 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to  
 97185 examine the next file in the argument list, but the final exit status shall be affected. If an error is  
 97186 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file  
 97187 in the argument list, but the final exit status shall be affected. If an error is encountered accessing  
 97188 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not  
 97189 be affected.

97190 **APPLICATION USAGE**

97191 When the standard output is not a terminal, only the **-s** filter-modification option is effective.  
 97192 This is based on historical practice. For example, a typical implementation of *man* pipes its  
 97193 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to  
 97194 *lp*, however, it is undesirable for this squeezing to happen.

97195 **EXAMPLES**

97196 The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

97197 *more -p G file1 file2*

97198 Examine each file starting with its last screenful.

97199 *more -p 100 file1 file2*

97200 Examine each file starting with line 100 in the current position (usually the third line, so line  
 97201 98 would be the first line written).

97202 *more -p /100 file1 file2*

97203 Examine each file starting with the first line containing the string "100" in the current  
 97204 position

## 97205 RATIONALE

97206 The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the  
 97207 POSIX file display program since it is more widely available than either the public-domain  
 97208 program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the  
 97209 features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use  
 97210 and has become more amenable for *vi* users. Several features originally derived from various file  
 97211 editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-2008 as they have  
 97212 proved extremely popular with users.

97213 There are inconsistencies between *more* and *vi* that result from historical practice. For example,  
 97214 the single-character commands **h**, **f**, **b**, and <space> are screen movers in *more*, but cursor  
 97215 movers in *vi*. These inconsistencies were maintained because the cursor movements are not  
 97216 applicable to *more* and the powerful functionality achieved without the use of the control key  
 97217 justifies the differences.

97218 The tags interface has been included in a program that is not a text editor because it promotes  
 97219 another degree of consistent operation with *vi*. It is conceivable that the paging environment of  
 97220 *more* would be superior for browsing source code files in some circumstances.

97221 The operating mode referred to for block-mode terminals effectively adds a <newline> to each  
 97222 Synopsis line that currently has none. So, for example, **d**<newline> would page one screenful.  
 97223 The mode could be triggered by a command line option, environment variable, or some other  
 97224 method. The details are not imposed by this volume of POSIX.1-2008 because there are so few  
 97225 systems known to support such terminals. Nevertheless, it was considered that all systems  
 97226 should be able to support *more* given the exception cited for this small community of terminals  
 97227 because, in comparison to *vi*, the cursor movements are few and the command set relatively  
 97228 amenable to the optional <newline> characters.

97229 Some versions of *more* provide a shell-escaping mechanism similar to the *ex !* command. The  
 97230 standard developers did not consider that this was necessary in a paginator, particularly given  
 97231 the wide acceptance of multiple window terminals and job control features. (They chose to  
 97232 retain such features in the editors and *mailx* because the shell interaction also gives an  
 97233 opportunity to modify the editing buffer, which is not applicable to *more*.)

97234 The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. The  
 97235 **+command** option is no longer specified by POSIX.1-2008 but may be present in some  
 97236 implementations. In early proposals, it took a *pattern* argument, but historical *less* provided the  
 97237 *more* general facility of a command. It would have been desirable to use the same **-c** as *ex* and *vi*,  
 97238 but the letter was already in use.

97239 The text stating “from a non-rewindable stream ... implementations may limit the amount of  
 97240 backwards motion supported” would allow an implementation that permitted no backwards  
 97241 motion beyond text already on the screen. It was not possible to require a minimum amount of  
 97242 backwards motion that would be effective for all conceivable device types. The implementation  
 97243 should allow the user to back up as far as possible, within device and reasonable memory  
 97244 allocation constraints.

97245 Historically, non-printable characters were displayed using the ARPA standard mappings,  
 97246 which are as follows:

- 97247 1. Printable characters are left alone.
- 97248 2. Control characters less than \177 are represented as followed by the character offset from  
 97249 the '@' character in the ASCII map; for example, \007 is represented as '@G'.

- 97250 3. \177 is represented as followed by ' ? ' .
- 97251 The display of characters having their eighth bit set was less standard. Existing implementations  
97252 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their  
97253 eighth bit set as the two characters "M-", followed by the seven-bit display as described  
97254 previously.) The latter probably has the best claim to historical practice because it was used with  
97255 the `-v` option of 4 BSD and 4 BSD-derived versions of the `cat` utility since 1980.
- 97256 No specific display format is required by POSIX.1-2008. Implementations are encouraged to  
97257 conform to historic practice in the absence of any strong reason to diverge.
- 97258 **FUTURE DIRECTIONS**
- 97259 None.
- 97260 **SEE ALSO**
- 97261 [Chapter 2](#) (on page 2297), [ctags](#), [ed](#), [ex](#), [vi](#)
- 97262 XBD [Chapter 8](#) (on page 173), [Section 9.2](#) (on page 182), [Section 9.3](#) (on page 183), [Section 12.2](#)  
97263 (on page 215)
- 97264 **CHANGE HISTORY**
- 97265 First released in Issue 4.
- 97266 **Issue 5**
- 97267 The FUTURE DIRECTIONS section is added.
- 97268 **Issue 6**
- 97269 This utility is marked as part of the User Portability Utilities option.
- 97270 The obsolescent SYNOPSIS is removed.
- 97271 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 97272
  - Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
  - The `more` utility should be able to handle underlined and emboldened displays of  
97273 characters that are wider than a single column position.
- 97274
- 97275 **Issue 7**
- 97276 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
97277 as an option delimiter in the OPTIONS section.
- 97278 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97279 **NAME**

97280 mv — move files

97281 **SYNOPSIS**97282 mv [-if] *source\_file target\_file*97283 mv [-if] *source\_file... target\_dir*97284 **DESCRIPTION**

97285 In the first synopsis form, the *mv* utility shall move the file named by the *source\_file* operand to  
 97286 the destination specified by the *target\_file*. This first synopsis form is assumed when the final  
 97287 operand does not name an existing directory and is not a symbolic link referring to an existing  
 97288 directory. In this case, if *target\_file* ends with a trailing <slash> character, *mv* shall treat this as an  
 97289 error and no *source\_file* operands will be processed.

97290 In the second synopsis form, *mv* shall move each file named by a *source\_file* operand to a  
 97291 destination file in the existing directory named by the *target\_dir* operand, or referenced if  
 97292 *target\_dir* is a symbolic link referring to an existing directory. The destination path for each  
 97293 *source\_file* shall be the concatenation of the target directory, a single <slash> character if the  
 97294 target did not end in a <slash>, and the last pathname component of the *source\_file*. This second  
 97295 form is assumed when the final operand names an existing directory.

97296 If any operand specifies an existing file of a type not specified by the System Interfaces volume  
 97297 of POSIX.1-2008, the behavior is implementation-defined.

97298 For each *source\_file* the following steps shall be taken.

- 97299 1. If the destination path exists, the *-f* option is not specified, and either of the following  
 97300 conditions is true:
  - 97301 a. The permissions of the destination path do not permit writing and the standard  
 97302 input is a terminal.
  - 97303 b. The *-i* option is specified.

97304 the *mv* utility shall write a prompt to standard error and read a line from standard input.  
 97305 If the response is not affirmative, *mv* shall do nothing more with the current *source\_file*  
 97306 and go on to any remaining *source\_files*.

- 97307 2. If the *source\_file* operand and destination path name the same existing file, then the  
 97308 destination path shall not be removed, and one of the following shall occur:
  - 97309 a. No change is made to *source\_file*, no error occurs, and no diagnostic is issued.
  - 97310 b. No change is made to *source\_file*, a diagnostic is issued to standard error  
 97311 identifying the two names, and the exit status is affected.
  - 97312 c. If the *source\_file* operand and destination path name distinct directory entries, then  
 97313 the *source\_file* operand is removed, no error occurs, and no diagnostic is issued.

97314 The *mv* utility shall do nothing more with the current *source\_file*, and go on to any  
 97315 remaining *source\_files*.

- 97316 3. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the  
 97317 System Interfaces volume of POSIX.1-2008, called with the following arguments:
  - 97318 a. The *source\_file* operand is used as the *old* argument.
  - 97319 b. The destination path is used as the *new* argument.

97320 If this succeeds, *mv* shall do nothing more with the current *source\_file* and go on to any  
 97321 remaining *source\_files*. If this fails for any reasons other than those described for the *errno*

- 97322 [EXDEV] in the System Interfaces volume of POSIX.1-2008, *mv* shall write a diagnostic  
 97323 message to standard error, do nothing more with the current *source\_file*, and go on to any  
 97324 remaining *source\_files*.
- 97325 4. If the destination path exists, and it is a file of type directory and *source\_file* is not a file of  
 97326 type directory, or it is a file not of type directory and *source\_file* is a file of type directory,  
 97327 *mv* shall write a diagnostic message to standard error, do nothing more with the current  
 97328 *source\_file*, and go on to any remaining *source\_files*. If the destination path exists and was  
 97329 created by a previous step, it is unspecified whether this will be treated as an error or the  
 97330 destination path will be overwritten.
- 97331 5. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*  
 97332 shall write a diagnostic message to standard error, do nothing more with the current  
 97333 *source\_file*, and go on to any remaining *source\_files*.
- 97334 6. The file hierarchy rooted in *source\_file* shall be duplicated as a file hierarchy rooted in the  
 97335 destination path. If *source\_file* or any of the files below it in the hierarchy are symbolic  
 97336 links, the links themselves shall be duplicated, including their contents, rather than any  
 97337 files to which they refer. The following characteristics of each file in the file hierarchy  
 97338 shall be duplicated:
- 97339 • The time of last data modification and time of last access
  - 97340 • The user ID and group ID
  - 97341 • The file mode
- 97342 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode  
 97343 bits S\_ISUID and S\_ISGID shall not be duplicated.
- 97344 When files are duplicated to another file system, the implementation may require that the  
 97345 process invoking *mv* has read access to each file being duplicated.
- 97346 If files being duplicated to another file system have hard links to other files, it is  
 97347 unspecified whether the files copied to the new file system have the hard links preserved  
 97348 or separate copies are created for the linked files.
- 97349 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic  
 97350 message to standard error, do nothing more with the current *source\_file*, and go on to any  
 97351 remaining *source\_files*.
- 97352 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic  
 97353 message to standard error, but this failure shall not cause *mv* to modify its exit status.
- 97354 7. The file hierarchy rooted in *source\_file* shall be removed. If this fails for any reason, *mv*  
 97355 shall write a diagnostic message to the standard error, do nothing more with the current  
 97356 *source\_file*, and go on to any remaining *source\_files*.

**OPTIONS**

- 97357 The *mv* utility shall conform to XBD Section 12.2 (on page 215).
- 97358
- 97359 The following options shall be supported:
- 97360 **-f** Do not prompt for confirmation if the destination path exists. Any previous  
 97361 occurrence of the **-i** option is ignored.
- 97362 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of  
 97363 the **-f** option is ignored.
- 97364 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option

97365 specified shall determine the behavior of *mv*.

#### 97366 OPERANDS

97367 The following operands shall be supported:

97368 *source\_file* A pathname of a file or directory to be moved.

97369 *target\_file* A new pathname for the file or directory being moved.

97370 *target\_dir* A pathname of an existing directory into which to move the input files.

#### 97371 STDIN

97372 The standard input shall be used to read an input line in response to each prompt specified in  
97373 the STDERR section. Otherwise, the standard input shall not be used.

#### 97374 INPUT FILES

97375 The input files specified by each *source\_file* operand can be of any file type.

#### 97376 ENVIRONMENT VARIABLES

97377 The following environment variables shall affect the execution of *mv*:

97378 *LANG* Provide a default value for the internationalization variables that are unset or null.  
97379 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
97380 variables used to determine the values of locale categories.)

97381 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
97382 internationalization variables.

97383 *LC\_COLLATE*

97384 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
97385 character collating elements used in the extended regular expression defined for  
97386 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

97387 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
97388 characters (for example, single-byte as opposed to multi-byte characters in  
97389 arguments and input files), the behavior of character classes used in the extended  
97390 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
97391 category.

97392 *LC\_MESSAGES*

97393 Determine the locale used to process affirmative responses, and the locale used to  
97394 affect the format and contents of diagnostic messages and prompts written to  
97395 standard error.

97396 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 97397 ASYNCHRONOUS EVENTS

97398 Default.

#### 97399 STDOUT

97400 Not used.

#### 97401 STDERR

97402 Prompts shall be written to the standard error under the conditions specified in the  
97403 DESCRIPTION section. The prompts shall contain the destination pathname, but their format is  
97404 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic  
97405 messages.

97406 **OUTPUT FILES**

97407 The output files may be of any file type.

97408 **EXTENDED DESCRIPTION**

97409 None.

97410 **EXIT STATUS**

97411 The following exit values shall be returned:

97412 0 All input files were moved successfully.

97413 &gt;0 An error occurred.

97414 **CONSEQUENCES OF ERRORS**

97415 If the copying or removal of *source\_file* is prematurely terminated by a signal or error, *mv* may  
 97416 leave a partial copy of *source\_file* at the source or destination. The *mv* utility shall not modify  
 97417 both *source\_file* and the destination path simultaneously; termination at any point shall leave  
 97418 either *source\_file* or the destination path complete.

97419 **APPLICATION USAGE**

97420 Some implementations mark for update the last file status change timestamp of renamed files  
 97421 and some do not. Applications which make use of the last file status change timestamp may  
 97422 behave differently with respect to renamed files unless they are designed to allow for either  
 97423 behavior.

97424 The specification ensures that *mv a a* will not alter the contents of file *a*, and allows the  
 97425 implementation to issue an error that a file cannot be moved onto itself. Likewise, when *a* and *b*  
 97426 are hard links to the same file, *mv a b* will not alter *b*, but if a diagnostic is not issued, then it is  
 97427 unspecified whether *a* is left untouched (as it would be by the *rename()* function) or unlinked  
 97428 (reducing the link count of *b*).

97429 **EXAMPLES**

97430 If the current directory contains only files *a* (of any type defined by the System Interfaces  
 97431 volume of POSIX.1-2008), *b* (also of any type), and a directory *c*:

97432 `mv a b c`97433 `mv c d`97434 results with the original files *a* and *b* residing in the directory *d* in the current directory.97435 **RATIONALE**

97436 Early proposals diverged from the SVID and BSD historical practice in that they required that  
 97437 when the destination path exists, the `-f` option is not specified, and input is not a terminal, *mv*  
 97438 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It  
 97439 should be noted that this is consistent with the *rename()* function defined in the System  
 97440 Interfaces volume of POSIX.1-2008, which does not require write permission on the target.

97441 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for  
 97442 confirmation, should be interpreted in the following manner:

```
97443 if (exists AND (NOT f_option) AND
97444     ((not_writable AND input_is_terminal) OR i_option))
```

97445 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally  
 97446 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv*  
 97447 deletes all existing destination paths without prompting, even when `-i` is specified; this is  
 97448 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when  
 97449 the file is unwritable and the standard input is not a terminal. The standard developers decided  
 97450 that use of `-i` is a request for interaction, so when the destination path exists, the utility takes

97451 instructions from whatever responds to standard input.

97452 The *rename()* function is able to move directories within the same file system. Some historical  
97453 versions of *mv* have been able to move directories, but not to a different file system. The  
97454 standard developers considered that this was an annoying inconsistency, so this volume of  
97455 POSIX.1-2008 requires directories to be able to be moved even across file systems. There is no **-R**  
97456 option to confirm that moving a directory is actually intended, since such an option was not  
97457 required for moving directories in historical practice. Requiring the application to specify it  
97458 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the  
97459 *rename()* function were preserved as much as possible. For example, *mv* is not permitted to  
97460 “rename” files to or from directories, even though they might be empty and removable.

97461 Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to  
97462 duplicate any file characteristics when moving a file across file systems, nor did they write a  
97463 diagnostic message for the user. The former behavior has been preserved to prevent scripts from  
97464 breaking; a diagnostic message is now required, however, so that users are alerted that the file  
97465 characteristics have changed.

97466 The exact format of the interactive prompts is unspecified. Only the general nature of the  
97467 contents of prompts are specified because implementations may desire more descriptive  
97468 prompts than those used on historical implementations. Therefore, an application not using the  
97469 **-f** option or using the **-i** option relies on the system to provide the most suitable dialog directly  
97470 with the user, based on the behavior specified.

97471 When *mv* is dealing with a single file system and *source\_file* is a symbolic link, the link itself is  
97472 moved as a consequence of the dependence on the *rename()* functionality, per the  
97473 DESCRIPTION. Across file systems, this has to be made explicit.

#### 97474 FUTURE DIRECTIONS

97475 None.

#### 97476 SEE ALSO

97477 *cp*, *ln*

97478 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

97479 XSH *rename()*

#### 97480 CHANGE HISTORY

97481 First released in Issue 2.

#### 97482 Issue 6

97483 The *mv* utility is changed to describe processing of symbolic links as specified in the  
97484 IEEE P1003.2b draft standard.

97485 The APPLICATION USAGE section is added.

#### 97486 Issue 7

97487 Austin Group Interpretation 1003.1-2001 #016 is applied.

97488 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
97489 *LC\_MESSAGES* environment variable.

97490 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

97491 SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.

97492 SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are  
97493 being duplicated to another file system while having hard links.

97494

Changes are made related to support for finegrained timestamps.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

97495 **NAME**

97496 newgrp — change to a new group

97497 **SYNOPSIS**97498 newgrp [-1] [*group*]97499 **DESCRIPTION**

97500 The *newgrp* utility shall create a new shell execution environment with a new real and effective  
 97501 group identification. Of the attributes listed in Section 2.12 (on page 2331), the new shell  
 97502 execution environment shall retain the working directory, file creation mask, and exported  
 97503 variables from the previous environment (that is, open files, traps, unexported variables, alias  
 97504 definitions, shell functions, and *set* options may be lost). All other aspects of the process  
 97505 environment that are preserved by the *exec* family of functions defined in the System Interfaces  
 97506 volume of POSIX.1-2008 shall also be preserved by *newgrp*; whether other aspects are preserved  
 97507 is unspecified.

97508 A failure to assign the new group identifications (for example, for security or password-related  
 97509 reasons) shall not prevent the new shell execution environment from being created.

97510 The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 97511 • On systems where the effective group ID is normally in the supplementary group list (or  
 97512 whenever the old effective group ID actually is in the supplementary group list):
  - 97513 — If the new effective group ID is also in the supplementary group list, *newgrp* shall  
 97514 change the effective group ID.
  - 97515 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add  
 97516 the new effective group ID to the list, if there is room to add it.
- 97517 • On systems where the effective group ID is not normally in the supplementary group list  
 97518 (or whenever the old effective group ID is not in the supplementary group list):
  - 97519 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete  
 97520 it.
  - 97521 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if  
 97522 there is room.

97523 **Note:** The System Interfaces volume of POSIX.1-2008 does not specify whether the effective group ID  
 97524 of a process is included in its supplementary group list.

97525 With no operands, *newgrp* shall change the effective group back to the groups identified in the  
 97526 user's user entry, and shall set the list of supplementary groups to that set in the user's group  
 97527 database entries.

97528 If the first argument is '-', the results are unspecified.

97529 If a password is required for the specified group, and the user is not listed as a member of that  
 97530 group in the group database, the user shall be prompted to enter the correct password for that  
 97531 group. If the user is listed as a member of that group, no password shall be requested. If no  
 97532 password is required for the specified group, it is implementation-defined whether users not  
 97533 listed as members of that group can change to that group. Whether or not a password is  
 97534 required, implementation-defined system accounting or security mechanisms may impose  
 97535 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and  
 97536 suppress the changing of the group identification.

**newgrp**

Utilities

97537 **OPTIONS**

97538 The *newgrp* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified  
 97539 usage of ' - ' .

97540 The following option shall be supported:

97541 **-l** (The letter ell.) Change the environment to what would be expected if the user  
 97542 actually logged in again.

97543 **OPERANDS**

97544 The following operand shall be supported:

97545 *group* A group name from the group database or a non-negative numeric group ID.  
 97546 Specifies the group ID to which the real and effective group IDs shall be set. If  
 97547 *group* is a non-negative numeric string and exists in the group database as a group  
 97548 name (see *getgrnam()*), the numeric group ID associated with that group name  
 97549 shall be used as the group ID.

97550 **STDIN**

97551 Not used.

97552 **INPUT FILES**

97553 The file */dev/tty* shall be used to read a single line of text for password checking, when one is  
 97554 required.

97555 **ENVIRONMENT VARIABLES**

97556 The following environment variables shall affect the execution of *newgrp*:

97557 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 97558 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 97559 variables used to determine the values of locale categories.)

97560 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 97561 internationalization variables.

97562 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 97563 characters (for example, single-byte as opposed to multi-byte characters in  
 97564 arguments).

97565 *LC\_MESSAGES*

97566 Determine the locale that should be used to affect the format and contents of  
 97567 diagnostic messages written to standard error.

97568 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

97569 **ASYNCHRONOUS EVENTS**

97570 Default.

97571 **STDOUT**

97572 Not used.

97573 **STDERR**

97574 The standard error shall be used for diagnostic messages and a prompt string for a password, if  
 97575 one is required. Diagnostic messages may be written in cases where the exit status is not  
 97576 available. See the EXIT STATUS section.

97577 **OUTPUT FILES**

97578 None.

97579 **EXTENDED DESCRIPTION**

97580 None.

97581 **EXIT STATUS**

97582 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group  
 97583 identification was changed successfully, the exit status shall be the exit status of the shell.  
 97584 Otherwise, the following exit value shall be returned:

97585 &gt;0 An error occurred.

97586 **CONSEQUENCES OF ERRORS**

97587 The invoking shell may terminate.

97588 **APPLICATION USAGE**

97589 There is no convenient way to enter a password into the group database. Use of group  
 97590 passwords is not encouraged, because by their very nature they encourage poor security  
 97591 practices. Group passwords may disappear in the future.

97592 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with  
 97593 *newgrp*, which in turn overlays itself with a new shell after changing group. On some  
 97594 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

97595 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a  
 97596 useful interface for the support of applications.

97597 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it  
 97598 successfully invokes a new shell and the rest of the original shell script is bypassed when the  
 97599 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.  
 97600 But usage such as:

```
97601 newgrp foo
97602 echo $?
```

97603 is not useful because the new shell might not have access to any status *newgrp* may have  
 97604 generated (and most historical systems do not provide this status). A zero status echoed here  
 97605 does not necessarily indicate that the user has changed to the new group successfully. Following  
 97606 *newgrp* with the *id* command provides a portable means of determining whether the group  
 97607 change was successful or not.

97608 **EXAMPLES**

97609 None.

97610 **RATIONALE**

97611 Most historical implementations use one of the *exec* functions to implement the behavior of  
 97612 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected  
 97613 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*  
 97614 issue a diagnostic message to tell the user that the environment changed, it would be  
 97615 inappropriate to require this change to some historical implementations.

97616 The password mechanism is allowed in the group database, but how this would be  
 97617 implemented is not specified.

97618 The *newgrp* utility was retained in this volume of POSIX.1-2008, even given the existence of the  
 97619 multiple group permissions feature in the System Interfaces volume of POSIX.1-2008, for several  
 97620 reasons. First, in some implementations, the group ownership of a newly created file is  
 97621 determined by the group of the directory in which the file is created, as allowed by the System

**newgrp**

Utilities

- 97622 Interfaces volume of POSIX.1-2008; on other implementations, the group ownership of a newly  
97623 created file is determined by the effective group ID. On implementations of the latter type,  
97624 *newgrp* allows files to be created with a specific group ownership. Finally, many  
97625 implementations use the real group ID in accounting, and on such systems, *newgrp* allows the  
97626 accounting identity of the user to be changed.
- 97627 **FUTURE DIRECTIONS**
- 97628 None.
- 97629 **SEE ALSO**
- 97630 [Chapter 2](#) (on page 2297), *sh*
- 97631 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 97632 XSH *exec*, *getgrnam()*
- 97633 **CHANGE HISTORY**
- 97634 First released in Issue 2.
- 97635 **Issue 6**
- 97636 This utility is marked as part of the User Portability Utilities option.
- 97637 The obsolescent SYNOPSIS is removed.
- 97638 The text describing supplemental groups is no longer conditional on {NGROUPS\_MAX} being  
97639 greater than 1. This is because {NGROUPS\_MAX} now has a minimum value of 8. This is a FIPS  
97640 requirement.
- 97641 **Issue 7**
- 97642 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
97643 argument is '-'.  
97644 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 97645 The *newgrp* utility is moved from the User Portability Utilities option to the Base. User  
97646 Portability Utilities is now an option for interactive utilities.

97647 **NAME**

97648 nice — invoke a utility with an altered nice value

97649 **SYNOPSIS**97650 nice [-n *increment*] *utility* [*argument...*]97651 **DESCRIPTION**

97652 The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see  
 97653 XBD Section 3.239, on page 71). With no options, the executed utility shall be run with a nice  
 97654 value that is some implementation-defined quantity greater than or equal to the nice value of the  
 97655 current process. If the user lacks appropriate privileges to affect the nice value in the requested  
 97656 manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be  
 97657 written to standard error, but this shall not prevent the invocation of *utility* or affect the exit  
 97658 status.

97659 **OPTIONS**97660 The *nice* utility shall conform to XBD Section 12.2 (on page 215).

97661 The following option is supported:

97662 **-n *increment*** A positive or negative decimal integer which shall have the same effect on the  
 97663 execution of the utility as if the utility had called the *nice()* function with the  
 97664 numeric value of the *increment* option-argument.

97665 **OPERANDS**

97666 The following operands shall be supported:

97667 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the  
 97668 special built-in utilities in Section 2.14 (on page 2334), the results are undefined.

97669 *argument* Any string to be supplied as an argument when invoking the utility named by the  
 97670 *utility* operand.

97671 **STDIN**

97672 Not used.

97673 **INPUT FILES**

97674 None.

97675 **ENVIRONMENT VARIABLES**97676 The following environment variables shall affect the execution of *nice*:

97677 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 97678 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 97679 variables used to determine the values of locale categories.)

97680 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 97681 internationalization variables.

97682 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 97683 characters (for example, single-byte as opposed to multi-byte characters in  
 97684 arguments).

97685 **LC\_MESSAGES**

97686 Determine the locale that should be used to affect the format and contents of  
 97687 diagnostic messages written to standard error.

97688 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.



97732 any other reason.

97733 **EXAMPLES**

97734 None.

97735 **RATIONALE**

97736 The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The  
 97737 command *nice -x utility*, for example, would be treated the same as the command *nice -1*  
 97738 *utility*. If the user does not have appropriate privileges, this results in a “permission denied”  
 97739 error. This is considered a bug.

97740 When a user without appropriate privileges gives a negative *increment*, System V treats it like  
 97741 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not  
 97742 run the utility. The standard specifies the System V behavior together with an optional BSD-style  
 97743 “permission denied” message.

97744 The C shell has a built-in version of *nice* that has a different interface from the one described in  
 97745 this volume of POSIX.1-2008.

97746 The term “utility” is used, rather than “command”, to highlight the fact that shell compound  
 97747 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.  
 97748 However, “utility” includes user application programs and shell scripts, not just utilities defined  
 97749 in this volume of POSIX.1-2008.

97750 Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the  
 97751 default nice value being the midpoint of that range. By default, they raise the nice value of the  
 97752 executed utility by 10.

97753 Some historical documentation states that the *increment* value must be within a fixed range. This  
 97754 is misleading; the valid *increment* values on any invocation are determined by the current  
 97755 process nice value, which is not always the default.

97756 The definition of nice value is not intended to suggest that all processes in a system have  
 97757 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
 97758 System Interfaces volume of POSIX.1-2008 make the notion of a single underlying priority for all  
 97759 scheduling policies problematic. Some implementations may implement the *nice*-related features  
 97760 to affect all processes on the system, others to affect just the general time-sharing activities  
 97761 implied by this volume of POSIX.1-2008, and others may have no effect at all. Because of the use  
 97762 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are  
 97763 possible.

97764 Earlier versions of this standard allowed a *-increment* option. This form is no longer specified by  
 97765 POSIX.1-2008 but may be present in some implementations.

97766 **FUTURE DIRECTIONS**

97767 None.

97768 **SEE ALSO**

97769 Chapter 2 (on page 2297), *renice*

97770 XBD Section 3.239 (on page 71), Chapter 8 (on page 173), Section 12.2 (on page 215)

97771 XSH *nice()*

97772 **CHANGE HISTORY**

97773 First released in Issue 4.

**nice***Utilities*97774 **Issue 6**

97775 This utility is marked as part of the User Portability Utilities option.

97776 The obsolescent SYNOPSIS is removed.

97777 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of  
97778 RATIONALE that referred to text no longer in the standard.

97779 **Issue 7**

97780 Austin Group Interpretation 1003.1-2001 #027 is applied.

97781 SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION,  
97782 APPLICATION USAGE, and RATIONALE sections.

97783 The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability  
97784 Utilities is now an option for interactive utilities.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

97785 **NAME**

97786 nl — line numbering filter

97787 **SYNOPSIS**

```
97788 xSI nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
97789 [-n format] [-s sep] [-v startnum] [-w width] [file]
```

97790 **DESCRIPTION**

97791 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and  
 97792 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional  
 97793 functionality may be provided in accordance with the command options in effect.

97794 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at  
 97795 the start of each logical page. A logical page consists of a header, a body, and a footer section.  
 97796 Empty sections are valid. Different line numbering options are independently available for  
 97797 header, body, and footer (for example, no numbering of header and footer lines while  
 97798 numbering blank lines only in the body).

97799 The starts of logical page sections shall be signaled by input lines containing nothing but the  
 97800 following delimiter characters:

Line	Start of
\: \: \:	Header
\: \:	Body
\:	Footer

97805 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

97806 **OPTIONS**

97807 The *nl* utility shall conform to XBD Section 12.2 (on page 215). Only one file can be named.

97808 The following options shall be supported:

97809 **-b *type*** Specify which logical page body lines shall be numbered. Recognized *types* and  
 97810 their meaning are:

97811 **a** Number all lines.

97812 **t** Number only non-empty lines.

97813 **n** No line numbering.

97814 **pstring** Number only lines that contain the basic regular expression specified in  
 97815 *string*.

97816 The default *type* for logical page body shall be **t** (text lines numbered).

97817 **-d *delim*** Specify the delimiter characters that indicate the start of a logical page section.  
 97818 These can be changed from the default characters "\:" to two user-specified  
 97819 characters. If only one character is entered, the second character shall remain the  
 97820 default character ' : ' .

97821 **-f *type*** Specify the same as **b *type*** except for footer. The default for logical page footer shall  
 97822 be **n** (no lines numbered).

97823 **-h *type*** Specify the same as **b *type*** except for header. The default *type* for logical page  
 97824 header shall be **n** (no lines numbered).

97825	<b>-i</b> <i>incr</i>	Specify the increment value used to number logical page lines. The default shall be 1.
97826		
97827	<b>-l</b> <i>num</i>	Specify the number of blank lines to be considered as one. For example, <b>-l 2</b> results in only the second adjacent blank line being numbered (if the appropriate <b>-h a</b> , <b>-b a</b> , or <b>-f a</b> option is set). The default shall be 1.
97828		
97829		
97830	<b>-n</b> <i>format</i>	Specify the line numbering format. Recognized values are: <b>ln</b> , left justified, leading zeros suppressed; <b>rn</b> , right justified, leading zeros suppressed; <b>rz</b> , right justified, leading zeros kept. The default <i>format</i> shall be <b>rn</b> (right justified).
97831		
97832		
97833	<b>-p</b>	Specify that numbering should not be restarted at logical page delimiters.
97834	<b>-s</b> <i>sep</i>	Specify the characters used in separating the line number and the corresponding text line. The default <i>sep</i> shall be a <tab>.
97835		
97836	<b>-v</b> <i>startnum</i>	Specify the initial value used to number logical page lines. The default shall be 1.
97837	<b>-w</b> <i>width</i>	Specify the number of characters to be used for the line number. The default <i>width</i> shall be 6.
97838		
97839	<b>OPERANDS</b>	
97840		The following operand shall be supported:
97841	<i>file</i>	A pathname of a text file to be line-numbered.
97842	<b>STDIN</b>	
97843		The standard input shall be used if no <i>file</i> operand is specified, and shall be used if the <i>file</i> operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.
97844		
97845		
97846	<b>INPUT FILES</b>	
97847		The input file shall be a text file.
97848	<b>ENVIRONMENT VARIABLES</b>	
97849		The following environment variables shall affect the execution of <i>nl</i> :
97850	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
97851		
97852		
97853	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
97854		
97855	<i>LC_COLLATE</i>	Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
97856		
97857		
97858	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, and for deciding which characters are in character class <b>graph</b> (for the <b>-b t</b> , <b>-f t</b> , and <b>-h t</b> options).
97859		
97860		
97861		
97862		
97863	<i>LC_MESSAGES</i>	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
97864		
97865		

97866 `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

#### 97867 **ASYNCHRONOUS EVENTS**

97868 Default.

#### 97869 **STDOUT**

97870 The standard output shall be a text file in the following format:

97871 `"%s%s%s", <line number>, <separator>, <input line>`

97872 where `<line number>` is one of the following numeric formats:

97873 `%6d` When the `rn` format is used (the default; see `-n`).

97874 `%06d` When the `rz` format is used.

97875 `%-6d` When the `ln` format is used.

97876 `<empty>` When line numbers are suppressed for a portion of the page; the `<separator>` is also suppressed.

97878 In the preceding list, the number 6 is the default width; the `-w` option can change this value.

#### 97879 **STDERR**

97880 The standard error shall be used only for diagnostic messages.

#### 97881 **OUTPUT FILES**

97882 None.

#### 97883 **EXTENDED DESCRIPTION**

97884 None.

#### 97885 **EXIT STATUS**

97886 The following exit values shall be returned:

97887 0 Successful completion.

97888 >0 An error occurred.

#### 97889 **CONSEQUENCES OF ERRORS**

97890 Default.

#### 97891 **APPLICATION USAGE**

97892 In using the `-d delim` option, care should be taken to escape characters that have special meaning to the command interpreter.

#### 97894 **EXAMPLES**

97895 The command:

97896 `nl -v 10 -i 10 -d \!+ file1`

97897 numbers `file1` starting at line number 10 with an increment of 10. The logical page delimiter is `"!+"`. Note that the `'!'` has to be escaped when using `cs`h as a command interpreter because of its history substitution syntax. For `ksh` and `sh` the escape is not necessary, but does not do any harm.

#### 97901 **RATIONALE**

97902 None.

97903 **FUTURE DIRECTIONS**

97904 None.

97905 **SEE ALSO**97906 *pr*97907 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)97908 **CHANGE HISTORY**

97909 First released in Issue 2.

97910 **Issue 5**97911 The option [*-f type*] is added to the SYNOPSIS. The option descriptions are presented in  
97912 alphabetic order. The description of *-bt* is changed to “Number only non-empty lines”.97913 **Issue 6**97914 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand  
97915 is removed.97916 **Issue 7**

97917 Austin Group Interpretation 1003.1-2001 #092 is applied.

97918 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

97919 **NAME**97920 nm — write the name list of an object file (**DEVELOPMENT**)97921 **SYNOPSIS**

```
97922 SD nm [-APv] [-g|-u] [-t format] file...
97923 XSI nm [-APv] [-efox] [-g|-u] [-t format] file...
```

97924 **DESCRIPTION**

97925 The *nm* utility shall display symbolic information appearing in the object file, executable file, or  
 97926 object-file library named by *file*. If no symbolic information is available for a valid input file, the  
 97927 *nm* utility shall report that fact, but not consider it an error condition.

97928 XSI The default base used when numeric values are written is unspecified. On XSI-conformant  
 97929 systems, it shall be decimal.

97930 **OPTIONS**97931 The *nm* utility shall conform to XBD Section 12.2 (on page 215).

97932 The following options shall be supported:

97933 **-A** Write the full pathname or library name of an object on each line.97934 XSI **-e** Write only external (global) and static symbol information.

97935 XSI **-f** Produce full output. Write redundant symbols (*.text*, *.data*, and *.bss*), normally  
 97936 suppressed.

97937 **-g** Write only external (global) symbol information.97938 XSI **-o** Write numeric values in octal (equivalent to **-t o**).97939 **-P** Write information in a portable output format, as specified in the STDOUT section.

97940 **-t *format*** Write each numeric value in the specified format. The format shall be dependent  
 97941 on the single character used as the *format* option-argument:

97942 XSI d The offset is written in decimal (default).

97943 o The offset is written in octal.

97944 x The offset is written in hexadecimal.

97945 **-u** Write only undefined symbols.97946 **-v** Sort output by value instead of alphabetically.97947 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).97948 **OPERANDS**

97949 The following operand shall be supported:

97950 *file* A pathname of an object file, executable file, or object-file library.97951 **STDIN**

97952 See the INPUT FILES section.

97953 **INPUT FILES**

97954 The input file shall be an object file, an object-file library whose format is the same as those  
 97955 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept  
 97956 additional implementation-defined object library formats for the input file.

97957 **ENVIRONMENT VARIABLES**97958 The following environment variables shall affect the execution of *nm*:

97959 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 97960 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 97961 variables used to determine the values of locale categories.)

97962 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 97963 internationalization variables.

97964 *LC\_COLLATE*  
 97965 Determine the locale for character collation information for the symbol-name and  
 97966 symbol-value collation sequences.

97967 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 97968 characters (for example, single-byte as opposed to multi-byte characters in  
 97969 arguments).

97970 *LC\_MESSAGES*  
 97971 Determine the locale that should be used to affect the format and contents of  
 97972 diagnostic messages written to standard error.

97973 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

97974 **ASYNCHRONOUS EVENTS**

97975 Default.

97976 **STDOUT**

97977 If symbolic information is present in the input files, then for each file or for each member of an  
 97978 archive, the *nm* utility shall write the following information to standard output. By default, the  
 97979 format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 97980 • Library or object name, if *-A* is specified
- 97981 • Symbol name
- 97982 • Symbol type, which shall either be one of the following single characters or an  
 97983 implementation-defined type represented by a single character:
  - 97984 A Global absolute symbol.
  - 97985 a Local absolute symbol.
  - 97986 B Global "bss" (that is, uninitialized data space) symbol.
  - 97987 b Local bss symbol.
  - 97988 D Global data symbol.
  - 97989 d Local data symbol.
  - 97990 T Global text symbol.
  - 97991 t Local text symbol.
  - 97992 U Undefined symbol.
- 97993 • Value of the symbol
- 97994 • The size associated with the symbol, if applicable

97995 This information may be supplemented by additional information specific to the  
 97996 implementation.

97997 If the **-P** option is specified, the previous information shall be displayed using the following  
 97998 portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,  
 97999 respectively:

98000 "%s%s %s %d %d\n", <library/object name>, <name>, <type>,  
 98001 <value>, <size>

98002 "%s%s %s %o %o\n", <library/object name>, <name>, <type>,  
 98003 <value>, <size>

98004 "%s%s %s %x %x\n", <library/object name>, <name>, <type>,  
 98005 <value>, <size>

98006 where <library/object name> shall be formatted as follows:

- 98007 • If **-A** is not specified, <library/object name> shall be an empty string.
- 98008 • If **-A** is specified and the corresponding *file* operand does not name a library:  
 98009 "%s: ", <file>
- 98010 • If **-A** is specified and the corresponding *file* operand names a library. In this case,  
 98011 <object file> shall name the object file in the library containing the symbol being described:  
 98012 "%s[%s]: ", <file>, <object file>

98013 If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is  
 98014 specified and it names a library, *nm* shall write a line identifying the object containing the  
 98015 following symbols before the lines containing those symbols, in the form:

- 98016 • If the corresponding *file* operand does not name a library:  
 98017 "%s:\n", <file>
- 98018 • If the corresponding *file* operand names a library; in this case, <object file> shall be the  
 98019 name of the file in the library containing the following symbols:  
 98020 "%s[%s]:\n", <file>, <object file>

98021 If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

#### 98022 **STDERR**

98023 The standard error shall be used only for diagnostic messages.

#### 98024 **OUTPUT FILES**

98025 None.

#### 98026 **EXTENDED DESCRIPTION**

98027 None.

#### 98028 **EXIT STATUS**

98029 The following exit values shall be returned:

- 98030 0 Successful completion.
- 98031 >0 An error occurred.

#### 98032 **CONSEQUENCES OF ERRORS**

98033 Default.

98034 **APPLICATION USAGE**

98035 Mechanisms for dynamic linking make this utility less meaningful when applied to an  
 98036 executable file because a dynamically linked executable may omit numerous library routines  
 98037 that would be found in a statically linked executable.

98038 **EXAMPLES**

98039 None.

98040 **RATIONALE**

98041 Historical implementations of *nm* have used different bases for numeric output and supplied  
 98042 different default types of symbols that were reported. The *-t format* option, similar to that used  
 98043 in *od* and *strings*, can be used to specify the numeric base; *-g* and *-u* can be used to restrict the  
 98044 amount of output or the types of symbols included in the output.

98045 The compromise of using *-t format versus* using *-d*, *-o*, and other similar options was necessary  
 98046 because of differences in the meaning of *-o* between implementations. The *-o* option from BSD  
 98047 has been provided here as *-A* to avoid confusion with the *-o* from System V (which has been  
 98048 provided here as *-t* and as *-o* on XSI-conformant systems).

98049 The option list was significantly reduced from that provided by historical implementations.

98050 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified  
 98051 default output.

98052 It was recognized that mechanisms for dynamic linking make this utility less meaningful when  
 98053 applied to an executable file (because a dynamically linked executable file may omit numerous  
 98054 library routines that would be found in a statically linked executable file), but the value of *nm*  
 98055 during software development was judged to outweigh other limitations.

98056 The default output format of *nm* is not specified because of differences in historical  
 98057 implementations. The *-P* option was added to allow some type of portable output format. After  
 98058 a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to  
 98059 create one that did not match the current format of any of these four systems. The format  
 98060 devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary  
 98061 depending on locale (because no English descriptions are included). All of the systems currently  
 98062 have the information available to use this format.

98063 The format given in *nm* STDOUT uses <space> characters between the fields, which may be any  
 98064 number of <blank> characters required to align the columns. The single-character types were  
 98065 selected to match historical practice, and the requirement that implementation additions also be  
 98066 single characters made parsing the information easier for shell scripts.

98067 **FUTURE DIRECTIONS**

98068 None.

98069 **SEE ALSO**

98070 *ar*, *c99*

98071 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

98072 **CHANGE HISTORY**

98073 First released in Issue 2.

98074 **Issue 6**

98075 This utility is marked as supported when both the User Portability Utilities option and the  
 98076 Software Development Utilities option are supported.

98077 **Issue 7**

98078 The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is  
98079 now an option for interactive utilities.

98080 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**nohup**

Utilities

98081 **NAME**

98082           nohup — invoke a utility immune to hangups

98083 **SYNOPSIS**98084           nohup *utility* [*argument...*]98085 **DESCRIPTION**

98086           The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied  
 98087           as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be  
 98088           set to be ignored.

98089           If standard input is associated with a terminal, the *nohup* utility may redirect standard input  
 98090           from an unspecified file.

98091           If the standard output is a terminal, all output written by the named *utility* to its standard output  
 98092           shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot  
 98093           be created or opened for appending, the output shall be appended to the end of the file  
 98094           **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be  
 98095           created or opened for appending, *utility* shall not be invoked. If a file is created, the file's  
 98096           permission bits shall be set to S\_IRUSR | S\_IWUSR.

98097           If standard error is a terminal and standard output is open but is not a terminal, all output  
 98098           written by the named utility to its standard error shall be redirected to the same open file  
 98099           description as the standard output. If standard error is a terminal and standard output either is a  
 98100           terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file  
 98101           as described above.

98102 **OPTIONS**

98103           None.

98104 **OPERANDS**

98105           The following operands shall be supported:

98106           *utility*           The name of a utility that is to be invoked. If the *utility* operand names any of the  
 98107           special built-in utilities in Section 2.14 (on page 2334), the results are undefined.

98108           *argument*       Any string to be supplied as an argument when invoking the utility named by the  
 98109           *utility* operand.

98110 **STDIN**

98111           Not used.

98112 **INPUT FILES**

98113           None.

98114 **ENVIRONMENT VARIABLES**98115           The following environment variables shall affect the execution of *nohup*:

98116           *HOME*           Determine the pathname of the user's home directory: if the output file **nohup.out**  
 98117           cannot be created in the current directory, the *nohup* utility shall use the directory  
 98118           named by *HOME* to create the file.

98119           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 98120           (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 98121           variables used to determine the values of locale categories.)

98122           *LC\_ALL*           If set to a non-empty string value, override the values of all the other  
 98123           internationalization variables.

- 98124 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 98125 characters (for example, single-byte as opposed to multi-byte characters in  
 98126 arguments).
- 98127 *LC\_MESSAGES*  
 98128 Determine the locale that should be used to affect the format and contents of  
 98129 diagnostic messages written to standard error.
- 98130 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 98131 *PATH* Determine the search path that is used to locate the utility to be invoked. See XBD  
 98132 Chapter 8 (on page 173).
- 98133 **ASYNCHRONOUS EVENTS**  
 98134 The *nohup* utility shall take the standard action for all signals except that *SIGHUP* shall be  
 98135 ignored.
- 98136 **STDOUT**  
 98137 If the standard output is not a terminal, the standard output of *nohup* shall be the standard  
 98138 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing  
 98139 shall be written to the standard output.
- 98140 **STDERR**  
 98141 If the standard output is a terminal, a message shall be written to the standard error, indicating  
 98142 the name of the file to which the output is being appended. The name of the file shall be either  
 98143 **nohup.out** or **\$HOME/nohup.out**.
- 98144 **OUTPUT FILES**  
 98145 Output written by the named utility is appended to the file **nohup.out** (or **\$HOME/nohup.out**),  
 98146 if the conditions hold as described in the **DESCRIPTION**.
- 98147 **EXTENDED DESCRIPTION**  
 98148 None.
- 98149 **EXIT STATUS**  
 98150 The following exit values shall be returned:
- 98151 126 The utility specified by *utility* was found but could not be invoked.
- 98152 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be  
 98153 found.
- 98154 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.
- 98155 **CONSEQUENCES OF ERRORS**  
 98156 Default.
- 98157 **APPLICATION USAGE**  
 98158 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 98159 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 98160 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 98161 used for other meanings; most utilities use small values for “normal error conditions” and the  
 98162 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 98163 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 98164 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 98165 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 98166 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 98167 any other reason.

**nohup**

Utilities

98168 **EXAMPLES**

98169 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by  
 98170 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
 98171 the *nohup* applies to everything in the file.

98172 Alternatively, the following command can be used to apply *nohup* to a complex command:

```
98173 nohup sh -c 'complex-command-line' </dev/null
```

98174 **RATIONALE**

98175 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails  
 98176 instead of trying to use **\$HOME/nohup.out**.

98177 The *cs* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this  
 98178 volume of POSIX.1-2008.

98179 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
 98180 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
 98181 includes user application programs and shell scripts, not just the standard utilities.

98182 Historical versions of the *nohup* utility use default file creation semantics. Some more recent  
 98183 versions use the permissions specified here as an added security precaution.

98184 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore  
 98185 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several  
 98186 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this  
 98187 volume of POSIX.1-2008.

98188 Historical versions of *nohup* did not affect standard input, but that causes problems in the  
 98189 common scenario where the user logs into a system, types the command:

```
98190 nohup make &
```

98191 at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session  
 98192 may not terminate for quite some time, since standard input remains open until *make* exits. To  
 98193 avoid this problem, POSIX.1-2008 allows implementations to redirect standard input if it is a  
 98194 terminal. Since the behavior is implementation-defined, portable applications that may run into  
 98195 the problem should redirect standard input themselves. For example, instead of:

```
98196 nohup make &
```

98197 an application can invoke:

```
98198 nohup make </dev/null &
```

98199 **FUTURE DIRECTIONS**

98200 None.

98201 **SEE ALSO**

98202 [Chapter 2](#) (on page 2297), *sh*

98203 [XBD Chapter 8](#) (on page 173)

98204 [XSH \*signal\(\)\*](#)

98205 **CHANGE HISTORY**

98206 First released in Issue 2.

98207 **Issue 7**  
98208

Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

## od

## 98209 NAME

98210 od — dump files in various formats

## 98211 SYNOPSIS

98212 od [-v] [-A *address\_base*] [-j *skip*] [-N *count*] [-t *type\_string*]...  
 98213 [*file*...]

98214 XSI od [-bcdosx] [*file*] [[+] *offset* [.][b]]

## 98215 DESCRIPTION

98216 The *od* utility shall write the contents of its input files to standard output in a user-specified  
 98217 format.

## 98218 OPTIONS

98219 The *od* utility shall conform to XBD Section 12.2 (on page 215), except that the order of  
 98220 XSI presentation of the **-t** options and the **-bcdosx** options is significant.

98221 The following options shall be supported:

98222 **-A *address\_base***

98223 Specify the input offset base. See the EXTENDED DESCRIPTION section. The  
 98224 application shall ensure that the *address\_base* option-argument is a character. The  
 98225 characters 'd', 'o', and 'x' specify that the offset base shall be written in  
 98226 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the  
 98227 offset shall not be written.

98228 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.

98229 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC\_CTYPE*  
 98230 category. Certain non-graphic characters appear as C escapes: "NUL=\0",  
 98231 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal  
 98232 numbers.

98233 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to  
 98234 **-t u2**.

98235 **-j *skip*** Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or  
 98236 seek past the first *skip* bytes in the concatenated input files. If the combined input is  
 98237 not at least *skip* bytes long, the *od* utility shall write a diagnostic message to  
 98238 standard error and exit with a non-zero exit status.

98239 By default, the *skip* option-argument shall be interpreted as a decimal number.  
 98240 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;  
 98241 otherwise, with a leading '0', the offset shall be interpreted as an octal number.  
 98242 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted  
 98243 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is  
 98244 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal  
 98245 digit.

98246 **-N *count*** Format no more than *count* bytes of input. By default, *count* shall be interpreted as  
 98247 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a  
 98248 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an  
 98249 octal number. If *count* bytes of input (after successfully skipping, if **-j *skip***  
 98250 is specified) are not available, it shall not be considered an error; the *od* utility shall  
 98251 format the input that is available.

98252	XSI	<b>-o</b>	Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to <b>-t o2</b> .
98253	XSI	<b>-s</b>	Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to
98254		<b>-t d2</b> .	
98255		<b>-t type_string</b>	
98256			Specify one or more output types. See the EXTENDED DESCRIPTION section. The
98257			application shall ensure that the <i>type_string</i> option-argument is a string specifying
98258			the types to be used when writing the input data. The string shall consist of the
98259			type specification characters <i>a</i> , <i>c</i> , <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> , specifying named character,
98260			character, signed decimal, floating point, octal, unsigned decimal, and
98261			hexadecimal, respectively. The type specification characters <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be
98262			followed by an optional unsigned decimal integer that specifies the number of
98263			bytes to be transformed by each instance of the output type. The type specification
98264			character <i>f</i> can be followed by an optional <i>F</i> , <i>D</i> , or <i>L</i> indicating that the conversion
98265			should be applied to an item of type <b>float</b> , <b>double</b> , or <b>long double</b> , respectively.
98266			The type specification characters <i>d</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be followed by an optional <i>C</i> , <i>S</i> ,
98267			<i>I</i> , or <i>L</i> indicating that the conversion should be applied to an item of type <b>char</b> ,
98268			<b>short</b> , <b>int</b> , or <b>long</b> , respectively. Multiple types can be concatenated within the
98269			same <i>type_string</i> and multiple <b>-t</b> options can be specified. Output lines shall be
98270			written for each type specified in the order in which the type specification
98271			characters are specified.
98272		<b>-v</b>	Write all input data. Without the <b>-v</b> option, any number of groups of output lines,
98273			which would be identical to the immediately preceding group of output lines
98274			(except for the byte offsets), shall be replaced with a line containing only an
98275			<asterisk> (' * ').
98276	XSI	<b>-x</b>	Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to <b>-t x2</b> .
98277	XSI		Multiple types can be specified by using multiple <b>-bcdostx</b> options. Output lines are written for
98278			each type specified in the order in which the types are specified.
98279		<b>OPERANDS</b>	
98280			The following operands shall be supported:
98281		<i>file</i>	A path name of a file to be read. If no <i>file</i> operands are specified, the standard input
98282			shall be used.
98283			If there are no more than two operands, none of the <b>-A</b> , <b>-j</b> , <b>-N</b> , <b>-t</b> , or <b>-v</b> options is
98284			specified, and either of the following is true: the first character of the last operand
98285			is a <plus-sign> (' + '), or there are two operands and the first character of the last
98286	XSI		operand is numeric; the last operand shall be interpreted as an offset operand on
98287			XSI-conformant systems. Under these conditions, the results are unspecified on
98288			systems that are not XSI-conformant systems.
98289	XSI	<b>[+]<i>offset</i>[.][b]</b>	The <i>offset</i> operand specifies the offset in the file where dumping is to commence.
98290			This operand is normally interpreted as octal bytes. If '.' is appended, the offset
98291			shall be interpreted in decimal. If 'b' is appended, the offset shall be interpreted
98292			in units of 512 bytes.
98293		<b>STDIN</b>	
98294			The standard input shall be used if no <i>file</i> operands are specified, and shall be used if a <i>file</i>
98295			operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
98296			the standard input shall not be used. See the INPUT FILES section.

98297 **INPUT FILES**

98298 The input files can be any file type.

98299 **ENVIRONMENT VARIABLES**98300 The following environment variables shall affect the execution of *od*:

98301 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 98302 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 98303 variables used to determine the values of locale categories.)

98304 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 98305 internationalization variables.

98306 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 98307 characters (for example, single-byte as opposed to multi-byte characters in  
 98308 arguments and input files).

98309 *LC\_MESSAGES*

98310 Determine the locale that should be used to affect the format and contents of  
 98311 diagnostic messages written to standard error.

98312 *LC\_NUMERIC*

98313 Determine the locale for selecting the radix character used when writing floating-  
 98314 point formatted output.

98315 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

98316 **ASYNCHRONOUS EVENTS**

98317 Default.

98318 **STDOUT**

98319 See the EXTENDED DESCRIPTION section.

98320 **STDERR**

98321 The standard error shall be used only for diagnostic messages.

98322 **OUTPUT FILES**

98323 None.

98324 **EXTENDED DESCRIPTION**

98325 The *od* utility shall copy sequentially each input file to standard output, transforming the input  
 98326 XSI data according to the output types specified by the *-t* option or the *-bcdox* options. If no  
 98327 output type is specified, the default output shall be as if *-t oS* had been specified.

98328 The number of bytes transformed by the output type specifier *c* may be variable depending on  
 98329 the *LC\_CTYPE* category.

98330 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds  
 98331 to the various C-language types as follows. If the *c99* compiler is present on the system, these  
 98332 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes  
 98333 may vary among systems that conform to POSIX.1-2008.

- 98334 • For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall
- 98335 correspond to the size of the underlying implementation's basic integer type. For these
- 98336 specifier characters, the implementation shall support values of the optional number of
- 98337 bytes to be converted corresponding to the number of bytes in the C-language types **char**,
- 98338 **short**, **int**, and **long**. These numbers can also be specified by an application as the
- 98339 characters '*C*', '*S*', '*I*', and '*L*', respectively. The implementation shall also support
- 98340 the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The

98341 implementation shall support the decimal value corresponding to the C-language type  
 98342 **long long**. The byte order used when interpreting numeric values is implementation-  
 98343 defined, but shall correspond to the order in which a constant of the corresponding type is  
 98344 stored in memory on the system.

98345 • For the type specifier character  $\mathfrak{f}$ , the default number of bytes shall correspond to the  
 98346 number of bytes in the underlying implementation's basic double precision floating-point  
 98347 data type. The implementation shall support values of the optional number of bytes to be  
 98348 converted corresponding to the number of bytes in the C-language types **float**, **double**,  
 98349 and **long double**. These numbers can also be specified by an application as the characters  
 98350 'F', 'D', and 'L', respectively.

98351 The type specifier character  $\mathfrak{a}$  specifies that bytes shall be interpreted as named characters from  
 98352 the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least  
 98353 significant seven bits of each byte shall be used for this type specification. Bytes with the values  
 98354 listed in the following table shall be written using the corresponding names for those characters.

98355 **Table 4-13** Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	If or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

98366 **Note:** The "\012" value may be written either as If or nl.

98367 The type specifier character  $\mathfrak{c}$  specifies that bytes shall be interpreted as characters specified by  
 98368 the current setting of the *LC\_CTYPE* locale category. Characters listed in the table in XBD  
 98369 Chapter 5 (on page 121) ('\ ', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as  
 98370 the corresponding escape sequences, except that <backslash> shall be written as a single  
 98371 <backslash> and a NUL shall be written as '\0'. Other non-printable characters shall be  
 98372 written as one three-digit octal number for each byte in the character. Printable multi-byte  
 98373 characters shall be written in the area corresponding to the first byte of the character; the two-  
 98374 character sequence "\*\*\*" shall be written in the area corresponding to each remaining byte in the  
 98375 character, as an indication that the character is continued. When either the *-j skip* or *-N count*  
 98376 option is specified along with the  $\mathfrak{c}$  type specifier, and this results in an attempt to start or finish  
 98377 in the middle of a multi-byte character, the result is implementation-defined.

98378 The input data shall be manipulated in blocks, where a block is defined as a multiple of the least  
 98379 common multiple of the number of bytes transformed by the specified output types. If the least  
 98380 common multiple is greater than 16, the results are unspecified. Each input block shall be  
 98381 written as transformed by each output type, one per written line, in the order that the output  
 98382 types were specified. If the input block size is larger than the number of bytes transformed by  
 98383 the output type, the output type shall sequentially transform the parts of the input block, and  
 98384 the output from each of the transformations shall be separated by one or more <blank>  
 98385 characters.

98386 If, as a result of the specification of the *-N* option or end-of-file being reached on the last input  
 98387 file, input data only partially satisfies an output type, the input shall be extended sufficiently

98388 with null bytes to write the last byte of the input.

98389 Unless **-A n** is specified, the first output line produced for each input block shall be preceded by  
 98390 the input offset, cumulative across input files, of the next byte to be written. The format of the  
 98391 input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the  
 98392 first character of the output line, and shall be followed by one or more <blank> characters. In  
 98393 addition, the offset of the byte following the last byte written shall be written after all the input  
 98394 data has been processed, but shall not be followed by any <blank> characters.

98395 If no **-A** option is specified, the input offset base is unspecified.

#### 98396 EXIT STATUS

98397 The following exit values shall be returned:

98398 0 All input files were processed successfully.

98399 >0 An error occurred.

#### 98400 CONSEQUENCES OF ERRORS

98401 Default.

#### 98402 APPLICATION USAGE

98403 XSI-conformant applications are warned not to use filenames starting with '+' or a first  
 98404 operand starting with a numeric character so that the old functionality can be maintained by  
 98405 implementations, unless they specify one of the **-A**, **-j** or **-N** options. To guarantee that one of  
 98406 these filenames is always interpreted as a filename, an application could always specify the  
 98407 address base format with the **-A** option.

#### 98408 EXAMPLES

98409 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as  
 98410 standard input to the command:

98411 `od -A d -t a`

98412 on an implementation using an input block size of 16 bytes, the standard output, independent of  
 98413 the current locale setting, would be similar to:

```

98414 00000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
98415 00000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
98416 00000032 sp ! " # $ % & ' ( ) * + , - . /
98417 00000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
98418 00000064 @ A B C D E F G H I J K L M N O
98419 00000080 P Q R S T U V W X Y Z [ \ ] ^ _
98420 00000096 ` a b c d e f g h i j k l m n o
98421 00000112 p q r s t u v w x y z { | } ~ del
98422 00000128
  
```

98423 Note that this volume of POSIX.1-2008 allows **nl** or **lf** to be used as the name for the  
 98424 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character  
 98425 **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and  
 98426 the POSIX locale character set symbolic name for the corresponding character is a <newline>.

98427 The command:

98428 `od -A o -t o2x2x -N 18`

98429 on a system with 32-bit words and an implementation using an input block size of 16 bytes  
 98430 could write 18 bytes in approximately the following format:

98431 0000000 032056 031440 041123 042040 052516 044530 020043 031464

```

98432          342e  3320  4253  4420  554e  4958  2023  3334
98433          342e3320      42534420      554e4958      20233334
98434 0000020 032472
98435          353a
98436          353a0000
98437 0000022

```

98438 The command:

```
98439 od -A d -t f -t o4 -t x4 -N 24 -j 0x15
```

98440 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point  
98441 format) would skip 21 bytes of input data and then write 24 bytes in approximately the  
98442 following format:

```

98443 0000000 1.0000000000000000e+00 1.5735000000000000e+01
98444 07774000000 0000000000 10013674121 35341217270
98445 3ff00000 00000000 402f3851 eb851eb8
98446 0000016 1.4066823000000000e+02
98447 10030312542 04370303230
98448 40619562 23e18698
98449 0000024

```

#### 98450 RATIONALE

98451 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently  
98452 *hexdump*. There were several objections to all of these based on the following reasons:

- 98453 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 98454 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 98455 • The *od* utility has been available on all historical implementations and there was no need to  
98456 create a new name for a utility so similar to the historical *od* utility.

98457 The original reasons for not standardizing historical *od* were also fairly widespread. Those  
98458 reasons are given below along with rationale explaining why the standard developers believe  
98459 that this version does not suffer from the indicated problem:

- 98460 • The BSD and System V versions of *od* have diverged, and the intersection of features  
98461 provided by both does not meet the needs of the user community. In fact, the System V  
98462 version only provides a mechanism for dumping octal bytes and **shorts**, signed and  
98463 unsigned decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability  
98464 to dump **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned  
98465 decimal, and hexadecimal **longs**. The version presented here provides more normalized  
98466 forms for dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned  
98467 decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as  
98468 current locale characters.
- 98469 • It would not be possible to come up with a compatible superset of the BSD and System V  
98470 flags that met the requirements of the standard developers. The historical default *od* output  
98471 is the specified default output of this utility. None of the option letters chosen for this  
98472 version of *od* conflict with any of the options to historical versions of *od*.
- 98473 • On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps  
98474 of **ints**, even in the BSD version. Because of the way options are named, the name space  
98475 could not be extended to solve these problems. This is why the **-t** option was added (with  
98476 type specifiers more closely matched to the *printf()* formats used in the rest of this volume  
98477 of POSIX.1-2008) and the optional field sizes were added to the *d*, *f*, *o*, *u*, and *x* type

98478 specifiers. It is also one of the reasons why the historical practice was not mandated as a  
 98479 required obsolescent form of *od*. (Although the old versions of *od* are not listed as an  
 98480 obsolescent form, implementations are urged to continue to recognize the older forms for  
 98481 several more years.) The *a*, *c*, *f*, *o*, and *x* types match the meaning of the corresponding  
 98482 format characters in the historical implementations of *od* except for the default sizes of the  
 98483 fields converted. The *d* format is signed in this volume of POSIX.1-2008 to match the  
 98484 *printf()* notation. (Historical versions of *od* used *d* as a synonym for *u* in this version. The  
 98485 System V implementation uses *s* for signed decimal; BSD uses *i* for signed decimal and *s*  
 98486 for null-terminated strings.) Other than *d* and *u*, all of the type specifiers match format  
 98487 characters in the historical BSD version of *od*.

98488 The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are  
 98489 used even though it is recognized that there may be zero or more than one compiler for the  
 98490 C language on an implementation and that they may use different sizes for some of these  
 98491 types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**,  
 98492 while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes  
 98493 **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the  
 98494 implementation for these types, corresponding to the values reported by invocations of the  
 98495 *getconf* utility when called with *system\_var* operands {UCHAR\_MAX}, {USHORT\_MAX},  
 98496 {UINT\_MAX}, and {ULONG\_MAX} for the types **char**, **short**, **int**, and **long**, respectively.  
 98497 There are similar constants required by the ISO C standard, but not required by the System  
 98498 Interfaces volume of POSIX.1-2008 or this volume of POSIX.1-2008. They are  
 98499 {FLT\_MANT\_DIG}, {DBL\_MANT\_DIG}, and {LDBL\_MANT\_DIG} for the types **float**,  
 98500 **double**, and **long double**, respectively. If the optional *c99* utility is provided by the  
 98501 implementation and used as specified by this volume of POSIX.1-2008, these are the sizes  
 98502 that would be provided. If an option is used that specifies different sizes for these types,  
 98503 there is no guarantee that the *od* utility is able to interpret binary data output by such a  
 98504 program correctly.

98505 This volume of POSIX.1-2008 requires that the numeric values of these lengths be  
 98506 recognized by the *od* utility and that symbolic forms also be recognized. Thus, a  
 98507 conforming application can always look at an array of **unsigned long** data elements using  
 98508 *od -t uL*.

- 98509 • The method of specifying the format for the address field based on specifying a starting  
 98510 offset in a file unnecessarily tied the two together. The *-A* option now specifies the address  
 98511 base and the *-S* option specifies a starting offset.
- 98512 • It would be difficult to break the dependence on US ASCII to achieve an internationalized  
 98513 utility. It does not seem to be any harder for *od* to dump characters in the current locale  
 98514 than it is for the *ed* or *sed* *l* commands. The *c* type specifier does this without difficulty and  
 98515 is completely compatible with the historical implementations of the *c* format character  
 98516 when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The  
 98517 *a* type specifier (from the BSD *a* format character) was left as a portable means to dump  
 98518 ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by  
 98519 *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard  
 98520 as a subset of their base codeset.

98521 The use of "\*" as an indication of continuation of a multi-byte character in *c* specifier output  
 98522 was chosen based on seeing an implementation that uses this method. The continuation bytes  
 98523 have to be marked in a way that is not ambiguous with another single-byte or multi-byte  
 98524 character.

98525 An early proposal used *-S* and *-n*, respectively, for the *-j* and *-N* options eventually selected.  
 98526 These were changed to avoid conflicts with historical implementations.

98527 The original standard specified `-t o2` as the default when no output type was given. This was  
 98528 changed to `-t oS` (the length of a **short**) to accommodate a supercomputer implementation that  
 98529 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not  
 98530 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at  
 98531 the same time to address an historical implementation that had no two-byte data types in its C  
 98532 compiler.

98533 The use of a basic integer data type is intended to allow the implementation to choose a word  
 98534 size commonly used by applications on that architecture.

98535 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
 98536 but this has been modified in this version.

#### 98537 FUTURE DIRECTIONS

98538 All option and operand interfaces marked XSI may be removed in a future version.

#### 98539 SEE ALSO

98540 *c99*, *sed*

98541 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

#### 98542 CHANGE HISTORY

98543 First released in Issue 2.

#### 98544 Issue 5

98545 In the description of the `-c` option, the phrase “This is equivalent to `-t c.`” is deleted.

98546 The FUTURE DIRECTIONS section is modified.

#### 98547 Issue 6

98548 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the  
 98549 revisions in the IEEE P1003.2b draft standard.

98550 The normative text is reworded to avoid use of the term “must” for application requirements.

98551 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples  
 98552 which used an undefined `-n` option, which should have been `-N`.

98553 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing  
 98554 behavior on systems with bytes consisting of more than eight bits.

#### 98555 Issue 7

98556 Austin Group Interpretation 1003.1-2001 #092 is applied.

98557 SD5-XCU-ERN-37 is applied, updating the OPERANDS section.

98558 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**paste**

Utilities

98559 **NAME**98560 `paste` — merge corresponding or subsequent lines of files98561 **SYNOPSIS**98562 `paste [-s] [-d list] file...`98563 **DESCRIPTION**98564 The *paste* utility shall concatenate the corresponding lines of the given input files, and write the  
98565 resulting lines to standard output.98566 The default operation of *paste* shall concatenate the corresponding lines of the input files. The  
98567 <newline> of every line except the line from the last input file shall be replaced with a <tab>.98568 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall  
98569 behave as though empty lines were read from the files on which end-of-file was detected, unless  
98570 the `-s` option is specified.98571 **OPTIONS**98572 The *paste* utility shall conform to XBD Section 12.2 (on page 215).

98573 The following options shall be supported:

98574 `-d list` Unless a <backslash> character appears in *list*, each character in *list* is an element  
98575 specifying a delimiter character. If a <backslash> character appears in *list*, the  
98576 <backslash> character and one or more characters following it are an element  
98577 specifying a delimiter character as described below. These elements specify one or  
98578 more delimiters to use, instead of the default <tab>, to replace the <newline> of  
98579 the input lines. The elements in *list* shall be used circularly; that is, when the list is  
98580 exhausted the first element from the list is reused. When the `-s` option is specified:

- 98581
- The last <newline> in a file shall not be modified.
  - The delimiter shall be reset to the first element of *list* after each *file* operand is processed.

98584 When the `-s` option is not specified:

- 98585
- The <newline> characters in the file specified by the last *file* operand shall not be modified.
  - The delimiter shall be reset to the first element of *list* each time a line is processed from each file.

98589 If a <backslash> character appears in *list*, it and the character following it shall be  
98590 used to represent the following delimiter characters:98591 `\n` <newline>.98592 `\t` <tab>.98593 `\\` <backslash> character.98594 `\0` Empty string (not a null character). If '`\0`' is immediately followed by the  
98595 character '`x`', the character '`X`', or any character defined by the `LC_CTYPE`  
98596 **digit** keyword (see XBD Chapter 7, on page 135), the results are unspecified.

98597 If any other characters follow the &lt;backslash&gt;, the results are unspecified.

98598 `-s` Concatenate all of the lines of each separate input file in command line order. The  
98599 <newline> of every line except the last line in each input file shall be replaced with  
98600 the <tab>, unless otherwise specified by the `-d` option.

98601 **OPERANDS**

98602 The following operand shall be supported:

98603 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the  
 98604 standard input shall be used; the standard input shall be read one line at a time,  
 98605 circularly, for each instance of '-'. Implementations shall support pasting of at  
 98606 least 12 *file* operands.

98607 **STDIN**98608 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES  
98609 section.98610 **INPUT FILES**

98611 The input files shall be text files, except that line lengths shall be unlimited.

98612 **ENVIRONMENT VARIABLES**98613 The following environment variables shall affect the execution of *paste*:

98614 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 98615 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
 98616 used to determine the values of locale categories.)

98617 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 98618 internationalization variables.

98619 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 98620 characters (for example, single-byte as opposed to multi-byte characters in  
 98621 arguments and input files).

98622 *LC\_MESSAGES*98623 Determine the locale that should be used to affect the format and contents of  
98624 diagnostic messages written to standard error.98625 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.98626 **ASYNCHRONOUS EVENTS**

98627 Default.

98628 **STDOUT**98629 Concatenated lines of input files shall be separated by the <tab> (or other characters under the  
98630 control of the -d option) and terminated by a <newline>.98631 **STDERR**

98632 The standard error shall be used only for diagnostic messages.

98633 **OUTPUT FILES**

98634 None.

98635 **EXTENDED DESCRIPTION**

98636 None.

98637 **EXIT STATUS**

98638 The following exit values shall be returned:

98639 0 Successful completion.

98640 &gt;0 An error occurred.

98641 **CONSEQUENCES OF ERRORS**

98642 If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic  
 98643 message shall be written to standard error, but no output is written to standard output. If the `-s`  
 98644 option is specified, the *paste* utility shall provide the default behavior described in Section 1.4 (on  
 98645 page 2288).

98646 **APPLICATION USAGE**

98647 When the escape sequences of the *list* option-argument are used in a shell script, they must be  
 98648 quoted; otherwise, the shell treats the <backslash> as a special character.

98649 Conforming applications should only use the specific <backslash>-escaped delimiters presented  
 98650 in this volume of POSIX.1-2008. Historical implementations treat '`\x`', where '`x`' is not in this  
 98651 list, as '`x`', but future implementations are free to expand this list to recognize other common  
 98652 escapes similar to those accepted by *printf* and other standard utilities.

98653 Most of the standard utilities work on text files. The *cut* utility can be used to turn files with  
 98654 arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used  
 98655 to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
98656 cut -b 1-500 -n file > file1
98657 cut -b 501- -n file > file2
```

98658 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
 98659 contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in  
 98660 *file* that are longer than 500 + {LINE\_MAX} bytes. The original file can be recreated from **file1**  
 98661 and **file2** using the command:

```
98662 paste -d "\0" file1 file2 > file
```

98663 The commands:

```
98664 paste -d "\0" ...
98665 paste -d "" ...
```

98666 are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-2008 and  
 98667 may result in an error. The construct '`\0`' is used to mean "no separator" because historical  
 98668 versions of *paste* did not follow the syntax guidelines, and the command:

```
98669 paste -d "" ...
```

98670 could not be handled properly by *getopt*().

98671 **EXAMPLES**

98672 1. Write out a directory in four columns:

```
98673 ls | paste - - - -
```

98674 2. Combine pairs of lines from a file into single lines:

```
98675 paste -s -d "\t\n" file
```

98676 **RATIONALE**

98677 None.

98678 **FUTURE DIRECTIONS**

98679 None.

98680 **SEE ALSO**98681 [Section 1.4](#) (on page 2288), *cut*, *grep*, *pr*98682 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)98683 **CHANGE HISTORY**

98684 First released in Issue 2.

98685 **Issue 6**

98686 The normative text is reworded to avoid use of the term “must” for application requirements.

98687 **Issue 7**

98688 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**patch**

Utilities

98689 **NAME**

98690 patch — apply changes to files

98691 **SYNOPSIS**98692 patch [-blNR] [-c|-e|-n|-u] [-d *dir*] [-D *define*] [-i *patchfile*]  
98693 [-o *outfile*] [-p *num*] [-r *rejectfile*] [*file*]98694 **DESCRIPTION**98695 The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff)  
98696 listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*)  
98697 and apply those differences to a file. By default, *patch* shall read from the standard input.98698 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*,  
98699 *-e*, *-n*, or *-u* option.98700 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they  
98701 came from separate patch files. (In this case, the application shall ensure that the name of the  
98702 patch file is determinable for each *diff* listing.)98703 **OPTIONS**98704 The *patch* utility shall conform to XBD Section 12.2 (on page 215).

98705 The following options shall be supported:

- 98706 **-b** Save a copy of the original contents of each modified file, before the differences are  
98707 applied, in a file of the same name with the suffix **.orig** appended to it. If the file  
98708 already exists, it shall be overwritten; if multiple patches are applied to the same  
98709 file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option  
98710 is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig*  
98711 shall be created.
- 98712 **-c** Interpret the patch file as a copied context difference (the output of the utility *diff*  
98713 when the *-c* or *-C* options are specified).
- 98714 **-d dir** Change the current directory to *dir* before processing as described in the  
98715 EXTENDED DESCRIPTION section.
- 98716 **-D define** Mark changes with one of the following C preprocessor constructs:  
98717 `#ifdef define`  
98718 `...`  
98719 `#endif`  
98720 `#ifndef define`  
98721 `...`  
98722 `#endif`  
98723 optionally combined with the C preprocessor construct **#else**. If the patched file is  
98724 processed with the C preprocessor, where the macro *define* is defined, the output  
98725 shall contain the changes from the patch file; otherwise, the output shall not  
98726 contain the patches specified in the patch file.
- 98727 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.
- 98728 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather  
98729 than the standard input.
- 98730 **-l** (The letter ell.) Cause any sequence of <blank> characters in the difference script to  
98731 match any sequence of <blank> characters in the input file. Other characters shall  
98732 be matched exactly.



**patch**

- 98778 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.
- 98779 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
98780 characters (for example, single-byte as opposed to multi-byte characters in  
98781 arguments and input files), and the behavior of character classes used in the  
98782 extended regular expression defined for the **yesexpr** locale keyword in the  
98783 *LC\_MESSAGES* category.
- 98784 *LC\_MESSAGES*  
98785 Determine the locale used to process affirmative responses, and the locale used to  
98786 affect the format and contents of diagnostic messages and prompts written to  
98787 standard error.
- 98788 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 98789 *LC\_TIME* Determine the locale for recognizing the format of file timestamps written by the  
98790 *diff* utility in a context-difference input file.
- 98791 **ASYNCHRONOUS EVENTS**
- 98792 Default.
- 98793 **STDOUT**
- 98794 Not used.
- 98795 **STDERR**
- 98796 The standard error shall be used for diagnostic and informational messages.
- 98797 **OUTPUT FILES**
- 98798 The output of the *patch* utility, the save files (**.orig** suffixes), and the reject files (**.rej** suffixes) shall  
98799 be text files.
- 98800 **EXTENDED DESCRIPTION**
- 98801 A patch file may contain patching instructions for more than one file; filenames shall be  
98802 determined as specified in [Filename Determination](#) (on page 2997). When the **-b** option is  
98803 specified, for each patched file, the original shall be saved in a file of the same name with the  
98804 suffix **.orig** appended to it.
- 98805 For each patched file, a reject file may also be created as noted in [Patch Application](#) (on page  
98806 2997). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix  
98807 **.rej** to the original filename.
- 98808 **Patch File Format**
- 98809 The patch file shall contain zero or more lines of header information followed by one or more  
98810 patches. Each patch shall contain zero or more lines of filename identification in the format  
98811 produced by the **-c**, **-C**, **-u**, or **-U** options of the *diff* utility, and one or more sets of *diff* output,  
98812 which are customarily called *hunks*.
- 98813 The *patch* utility shall recognize the following expression in the header information:
- 98814 **Index:** *pathname*  
98815 The file to be patched is named *pathname*.
- 98816 If all lines (including headers) within a patch begin with the same leading sequence of <blank>  
98817 characters, the *patch* utility shall remove this sequence before proceeding. Within each patch, if  
98818 the type of difference is common context, the *patch* utility shall recognize the following  
98819 expressions:

98820       \*\*\* *filename timestamp*  
98821       The patches arose from *filename*.

98822       --- *filename timestamp*  
98823       The patches should be applied to *filename*.

98824       If the type of difference is unified context, the *patch* utility shall recognize the following  
98825       expressions:

98826       --- *filename timestamp*  
98827       The patches arose from *filename*.

98828       +++ *filename timestamp*  
98829       The patches should be applied to *filename*.

98830       Each hunk within a patch shall be the *diff* output to change a line range within the original file.  
98831       The line numbers for successive hunks within a patch shall occur in ascending order.

### 98832       Filename Determination

98833       If no *file* operand is specified, *patch* shall perform the following steps to determine the filename  
98834       to use:

- 98835       1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as  
98836       specified by the **-p** option) from the filename on the line beginning with "\*\*\*" (if copied  
98837       context) or "---" (if unified context), then test for the existence of this file relative to the  
98838       current directory (or the directory specified with the **-d** option). If the file exists, the *patch*  
98839       utility shall use this filename.
- 98840       2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as  
98841       specified by the **-p** option) from the filename on the line beginning with "---" (if copied  
98842       context) or "+++" (if unified context), then test for the existence of this file relative to the  
98843       current directory (or the directory specified with the **-d** option). If the file exists, the *patch*  
98844       utility shall use this filename.
- 98845       3. If the header information contains a line beginning with the string **Index:**, the *patch* utility  
98846       shall delete pathname components (as specified by the **-p** option) from this line, then test  
98847       for the existence of this file relative to the current directory (or the directory specified  
98848       with the **-d** option). If the file exists, the *patch* utility shall use this filename.
- 98849       4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a *get -e*  
98850       *SCCS/s.filename* command to retrieve an editable version of the file. If the file exists, the  
98851       *patch* utility shall use this filename.
- 98852       5. The *patch* utility shall write a prompt to standard output and request a filename  
98853       interactively from the controlling terminal (for example, **/dev/tty**).

### 98854       Patch Application

98855       If the **-c**, **-e**, **-n**, or **-u** option is present, the *patch* utility shall interpret information within each  
98856       hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context  
98857       difference, respectively. In the absence of any of these options, the *patch* utility shall determine  
98858       the type of difference based on the format of information within the hunk.

98859       For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line  
98860       number at the beginning of the hunk, plus or minus any offset used in applying the previous  
98861       hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and  
98862       backwards at least 1 000 bytes for a set of lines that match the hunk context.

**patch**

98863 If no such place is found and it is a context difference, then another scan shall take place,  
 98864 ignoring the first and last line of context. If that fails, the first two and last two lines of context  
 98865 shall be ignored and another scan shall be made. Implementations may search more extensively  
 98866 for installation locations.

98867 If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected  
 98868 hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written  
 98869 in copied-context-difference format regardless of the format of the patch file. It is  
 98870 implementation-defined whether a rejected hunk that is a unified context difference is written in  
 98871 copied-context-difference format or in unified-context-difference format. If the input was a  
 98872 normal or *ed*-style difference, the reject file may contain differences with zero lines of context.  
 98873 The line numbers on the hunks in the reject file may be different from the line numbers in the  
 98874 patch file since they shall reflect the approximate locations for the failed hunks in the new file  
 98875 rather than the old one.

98876 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking  
 98877 the *ed* utility.

**EXIT STATUS**

98879 The following exit values shall be returned:

- 98880 0 Successful completion.
- 98881 1 One or more lines were written to a reject file.
- 98882 >1 An error occurred.

**CONSEQUENCES OF ERRORS**

98884 Patches that cannot be correctly placed in the file shall be written to a reject file.

**APPLICATION USAGE**

98886 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct  
 98887 the reverse operation.

98888 The **-p** option makes it possible to customize a patch file to local user directory structures  
 98889 without manually editing the patch file. For example, if the filename in the patch file was:

98890 `/curds/whey/src/blurfl/blurfl.c`

98891 Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

98892 `curds/whey/src/blurfl/blurfl.c`

98893 without the leading <slash>, **-p 4** gives:

98894 `blurfl/blurfl.c`

98895 and not specifying **-p** at all gives:

98896 `blurfl.c` .

**EXAMPLES**

98898 None.

**RATIONALE**

98900 Some of the functionality in historical *patch* implementations was not specified. The following  
 98901 documents those features present in historical implementations that have not been specified.

98902 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options  
 98903 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

98904 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility

98905 would search for the corresponding version information (the string specified in the header,  
98906 delimited by <blank> characters or the beginning or end of a line or the file) anywhere in the  
98907 original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to  
98908 standardize. For example, if:

98909 Prereq: 1.2

98910 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the  
98911 prerequisite.

98912 The following options were dropped from historical implementations of *patch* as insufficiently  
98913 useful to standardize:

98914 **-b** The **-b** option historically provided a method for changing the name extension of  
98915 the backup file from the default **.orig**. This option has been modified and retained  
98916 in this volume of POSIX.1-2008.

98917 **-F** The **-F** option specified the number of lines of a context diff to ignore when  
98918 searching for a place to install a patch.

98919 **-f** The **-f** option historically caused *patch* not to request additional information from  
98920 the user.

98921 **-r** The **-r** option historically provided a method of overriding the extension of the  
98922 reject file from the default **.rej**.

98923 **-s** The **-s** option historically caused *patch* to work silently unless an error occurred.

98924 **-x** The **-x** option historically set internal debugging flags.

98925 In some file system implementations, the saving of a **.orig** file may produce unwanted results. In  
98926 the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum  
98927 filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename  
98928 limit. It was suggested, due to some historical practice, that a <tilde> ('~') suffix be used  
98929 instead of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is  
98930 not obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more  
98931 understandable.

98932 The **-b** option has the opposite sense in some historical implementations—do not save the **.orig**  
98933 file. The default case here is not to save the files, making *patch* behave more consistently with the  
98934 other standard utilities.

98935 The **-w** option in early proposals was changed to **-l** to match historical practice.

98936 The **-N** option was included because without it, a non-interactive application cannot reject  
98937 previously applied patches. For example, if a user is piping the output of *diff* into the *patch*  
98938 utility, and the user only wants to patch a file to a newer version non-interactively, the **-N** option  
98939 is required.

98940 Changes to the **-l** option description were proposed to allow matching across <newline>  
98941 characters in addition to just <blank> characters. Since this is not historical practice, and since  
98942 some ambiguities could result, it is suggested that future developments in this area utilize  
98943 another option letter, such as **-L**.

98944 The **-u** option of GNU *patch* has been added, along with support for unified context formats.

**patch**98945 **FUTURE DIRECTIONS**

98946 None.

98947 **SEE ALSO**98948 *diff, ed*98949 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)98950 **CHANGE HISTORY**

98951 First released in Issue 4.

98952 **Issue 5**

98953 The FUTURE DIRECTIONS section is added.

98954 **Issue 6**

98955 This utility is marked as part of the User Portability Utilities option.

98956 The description of the **-D** option and the steps in [Filename Determination](#) (on page 2997) are changed to match historical practice as defined in the IEEE P1003.2b draft standard.

98958 The normative text is reworded to avoid use of the term “must” for application requirements.

98959 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the *patch* utility performs **ifdef** selection for the **-D** option.98961 **Issue 7**98962 The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

98964 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

98965 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the **-u** option.98966 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC\_MESSAGES* and *LC\_CTYPE* environment variables and adding the *LC\_COLLATE* environment variable.

98969 **NAME**

98970 pathchk — check pathnames

98971 **SYNOPSIS**98972 pathchk [-p] [-P] *pathname*...98973 **DESCRIPTION**

98974 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used  
 98975 to access or create a file without causing syntax errors) and portable (that is, no filename  
 98976 truncation results). More extensive portability checks are provided by the **-p** and **-P** options.

98977 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the  
 98978 underlying file system. A diagnostic shall be written for each *pathname* operand that

- 98979 • Is longer than {PATH\_MAX} bytes (see **Pathname Variable Values** in XBD Chapter 13 (on  
 98980 page 219), <limits.h>)
- 98981 • Contains any component longer than {NAME\_MAX} bytes in its containing directory
- 98982 • Contains any component in a directory that is not searchable
- 98983 • Contains any character in any component that is not valid in its containing directory

98984 The format of the diagnostic message is not specified, but shall indicate the error detected and  
 98985 the corresponding *pathname* operand.

98986 It shall not be considered an error if one or more components of a *pathname* operand do not exist  
 98987 as long as a file matching the pathname specified by the missing components could be created  
 98988 that does not violate any of the checks specified above.

98989 **OPTIONS**98990 The *pathchk* utility shall conform to XBD Section 12.2 (on page 215).

98991 The following option shall be supported:

- 98992 **-p** Instead of performing checks based on the underlying file system, write a  
 98993 diagnostic for each *pathname* operand that:
- 98994 • Is longer than {\_POSIX\_PATH\_MAX} bytes (see **Minimum Values** in XBD  
 98995 Chapter 13 (on page 219), <limits.h>)
  - 98996 • Contains any component longer than {\_POSIX\_NAME\_MAX} bytes
  - 98997 • Contains any character in any component that is not in the portable filename  
 98998 character set
- 98999 **-P** Write a diagnostic for each *pathname* operand that:
- 99000 • Contains a component whose first character is the <hyphen> character
  - 99001 • Is empty

99002 **OPERANDS**

99003 The following operand shall be supported:

99004 *pathname* A pathname to be checked.99005 **STDIN**

99006 Not used.

**pathchk**

Utilities

99007 **INPUT FILES**

99008 None.

99009 **ENVIRONMENT VARIABLES**99010 The following environment variables shall affect the execution of *pathchk*:

99011 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 99012 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
 99013 used to determine the values of locale categories.)

99014 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 99015 internationalization variables.

99016 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 99017 characters (for example, single-byte as opposed to multi-byte characters in  
 99018 arguments).

99019 *LC\_MESSAGES*

99020 Determine the locale that should be used to affect the format and contents of  
 99021 diagnostic messages written to standard error.

99022 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

99023 **ASYNCHRONOUS EVENTS**

99024 Default.

99025 **STDOUT**

99026 Not used.

99027 **STDERR**

99028 The standard error shall be used only for diagnostic messages.

99029 **OUTPUT FILES**

99030 None.

99031 **EXTENDED DESCRIPTION**

99032 None.

99033 **EXIT STATUS**

99034 The following exit values shall be returned:

99035 0 All *pathname* operands passed all of the checks.

99036 &gt;0 An error occurred.

99037 **CONSEQUENCES OF ERRORS**

99038 Default.

99039 **APPLICATION USAGE**

99040 The *test* utility can be used to determine whether a given pathname names an existing file; it  
 99041 does not, however, give any indication of whether or not any component of the pathname was  
 99042 truncated in a directory where the *\_POSIX\_NO\_TRUNC* feature is not in effect. The *pathchk*  
 99043 utility does not check for file existence; it performs checks to determine whether a pathname  
 99044 does exist or could be created with no pathname component truncation.

99045 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a  
 99046 file. As with all file creation semantics in the System Interfaces volume of POSIX.1-2008, it  
 99047 guarantees atomic creation, but still depends on applications to agree on conventions and  
 99048 cooperate on the use of files after they have been created.

99049 To verify that a pathname meets the requirements of filename portability, applications should

99050 use both the `-p` and `-P` options together.

#### 99051 EXAMPLES

99052 To verify that all pathnames in an imported data interchange archive are legitimate and  
99053 unambiguous on the current system:

```
99054 # This example assumes that no pathnames in the archive
99055 # contain <newline> characters.
99056 pax -f archive | sed -e 's/[^[[:alnum:]]/\\&/g' | xargs pathchk --
99057 if [ $? -eq 0 ]
99058 then
99059     pax -r -f archive
99060 else
99061     echo Investigate problems before importing files.
99062     exit 1
99063 fi
```

99064 To verify that all files in the current directory hierarchy could be moved to any system  
99065 conforming to the System Interfaces volume of POSIX.1-2008 that also supports the `pax` utility:

```
99066 find . -exec pathchk -p -P {} +
99067 if [ $? -eq 0 ]
99068 then
99069     pax -w -f ../archive .
99070 else
99071     echo Portable archive cannot be created.
99072     exit 1
99073 fi
```

99074 To verify that a user-supplied pathname names a readable file and that the application can create  
99075 a file extending the given path without truncation and without overwriting any existing file:

```
99076 case $- in
99077     *C*)   reset="";;
99078     *)     reset="set +C"
99079           set -C;
99080 esac
99081 test -r "$path" && pathchk "$path.out" &&
99082 rm "$path.out" > "$path.out"
99083 if [ $? -ne 0 ]; then
99084     printf "%s: %s not found or %s.out fails \
99085 creation checks.\n" $0 "$path" "$path"
99086     $reset      # Reset the noclobber option in case a trap
99087                # on EXIT depends on it.
99088     exit 1
99089 fi
99090 $reset
99091 PROCESSING < "$path" > "$path.out"
```

99092 The following assumptions are made in this example:

- 99093 1. **PROCESSING** represents the code that is used by the application to use `$path` once it is  
99094 verified that `$path.out` works as intended.

## pathchk

- 99095 2. The state of the *noclobber* option is unknown when this code is invoked and should be set  
 99096 on exit to the state it was in when this code was invoked. (The **reset** variable is used in  
 99097 this example to restore the initial state.)
- 99098 3. Note the usage of:
- 99099 `rm "$path.out" > "$path.out"`
- 99100 a. The *pathchk* command has already verified, at this point, that **\$path.out** is not  
 99101 truncated.
- 99102 b. With the *noclobber* option set, the shell verifies that **\$path.out** does not already exist  
 99103 before invoking *rm*.
- 99104 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application  
 99105 can create the file again in the **PROCESSING** step.
- 99106 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:  
 99107 `rm "$path.out" > "$path.out"`  
 99108 should be replaced with:  
 99109 `> "$path.out"`  
 99110 which verifies that the file did not already exist, but leaves **\$path.out** in place for  
 99111 use by **PROCESSING**.

## RATIONALE

99112 The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set*  
 99113 `-C(noclobber)` option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that  
 99114 appeared in early proposals. All of these utilities were attempts to solve several common  
 99115 problems:  
 99116

- 99117 • Verify the validity (for several different definitions of “valid”) of a pathname supplied by a  
 99118 user, generated by an application, or imported from an external source.
- 99119 • Atomically create a file
- 99120 • Perform various string handling functions to generate a temporary filename.

99121 The *create* utility, included in an early proposal, provided checking and atomic creation in a  
 99122 single invocation of the utility; these are orthogonal issues and need not be grouped into a single  
 99123 utility. Note that the *noclobber* option also provides a way of creating a lock for process  
 99124 synchronization; since it provides an atomic *create*, there is no race between a test for existence  
 99125 and the following creation if it did not exist.

99126 Having a function like *tmpnam()* in the ISO C standard is important in many high-level  
 99127 languages. The shell programming language, however, has built-in string manipulation  
 99128 facilities, making it very easy to construct temporary filenames. The names needed obviously  
 99129 depend on the application, but are frequently of a form similar to:

99130 `$TMPDIR/application_abbreviation$$ .suffix`

99131 In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop  
 99132 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as  
 99133 applications trying to use the same filename name space are cooperating on the use of files after  
 99134 they have been created.

99135 For historical purposes, `-p` does not check for the use of the <hyphen> character as the first  
 99136 character in a component of the pathname, or for an empty *pathname* operand.

99137 **FUTURE DIRECTIONS**

99138 None.

99139 **SEE ALSO**99140 [Section 2.7](#) (on page 2312), [set](#) (on page 2357), [test](#)99141 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215), [<limits.h>](#)99142 **CHANGE HISTORY**

99143 First released in Issue 4.

99144 **Issue 7**

99145 Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

99146 SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

**pax**

Utilities

99147 **NAME**

99148 pax — portable archive interchange

99149 **SYNOPSIS**99150 pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive] [-s replstr]...  
99151 [pattern...]99152 pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]... [-p string]...  
99153 [-s replstr]... [pattern...]99154 pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]] [-o options]...  
99155 [-s replstr]... [-x format] [file...]99156 pax -r -w [-diklntuvX] [-H|-L] [-o options]... [-p string]...  
99157 [-s replstr]... [file...] directory99158 **DESCRIPTION**99159 The *pax* utility shall read, write, and write lists of the members of archive files and copy  
99160 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.99161 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations  
99162 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,  
99163 corresponding respectively to the four forms shown in the SYNOPSIS section.99164 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of  
99165 the members of the archive file read from the standard input, with pathnames  
99166 matching the specified patterns, to standard output. If a named file is of type  
99167 directory, the file hierarchy rooted at that file shall be listed as well.99168 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of  
99169 the archive file read from the standard input, with pathnames matching the  
99170 specified patterns. If an extracted file is of type directory, the file hierarchy rooted  
99171 at that file shall be extracted as well. The extracted files shall be created performing  
99172 pathname resolution with the directory in which *pax* was invoked as the current  
99173 working directory.99174 If an attempt is made to extract a directory when the directory already exists, this  
99175 shall not be considered an error. If an attempt is made to extract a FIFO when the  
99176 FIFO already exists, this shall not be considered an error.99177 The ownership, access, and modification times, and file mode of the restored files  
99178 are discussed under the *-p* option.99179 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of  
99180 the *file* operands to the standard output in an archive format. If no *file* operands are  
99181 specified, a list of files to copy, one per line, shall be read from the standard input  
99182 and each entry in this list shall be processed as if it had been a *file* operand on the  
99183 command line. A file of type directory shall include all of the files in the file  
99184 hierarchy rooted at the file.99185 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands  
99186 to the destination directory.99187 If no *file* operands are specified, a list of files to copy, one per line, shall be read  
99188 from the standard input. A file of type directory shall include all of the files in the  
99189 file hierarchy rooted at the file.99190 The effect of the **copy** shall be as if the copied files were written to a *pax* format  
99191 archive file and then subsequently extracted, except that there may be hard links

99192 between the original and the copied files. If the destination directory is a  
 99193 subdirectory of one of the files to be copied, the results are unspecified. If the  
 99194 destination directory is a file of a type not defined by the System Interfaces volume  
 99195 of POSIX.1-2008, the results are implementation-defined; otherwise, it shall be an  
 99196 error for the file named by the *directory* operand not to exist, not be writable by the  
 99197 user, or not be a file of type directory.

99198 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,  
 99199 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces  
 99200 volume of POSIX.1-2008, called with the following arguments:

- 99201 • The intermediate directory used as the *path* argument
- 99202 • The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO as the *mode*  
 99203 argument

99204 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,  
 99205 *pax* shall write a diagnostic message to standard error for each one that did not match and exit  
 99206 with a non-zero exit status.

99207 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically  
 99208 detected on input. The default output archive format shall be implementation-defined.

99209 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-  
 99210 defined manner, what file to read or write as the next file.

99211 If the selected archive format supports the specification of linked files, it shall be an error if these  
 99212 files cannot be linked when the archive is extracted. For archive formats that do not store file  
 99213 contents with each name that causes a hard link, if the file that contains the data is not extracted  
 99214 during this *pax* session, either the data shall be restored from the original file, or a diagnostic  
 99215 message shall be displayed with the name of a file that can be used to extract the data. In  
 99216 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited  
 99217 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall  
 99218 write a diagnostic message to standard error and shall terminate.

## 99219 OPTIONS

99220 The *pax* utility shall conform to XBD Section 12.2 (on page 215), except that the order of  
 99221 presentation of the **-o**, **-p**, and **-s** options is significant.

99222 The following options shall be supported:

- 99223 **-r** Read an archive file from standard input.
- 99224 **-w** Write files to the standard output in the specified archive format.
- 99225 **-a** Append files to the end of the archive. It is implementation-defined which devices  
 99226 on the system support appending. Additional file formats unspecified by this  
 99227 volume of POSIX.1-2008 may impose restrictions on appending.
- 99228 **-b** *blocksize* Block the output at a positive decimal integer number of bytes per write to the  
 99229 archive file. Devices and archive formats may impose restrictions on blocking.  
 99230 Blocking shall be automatically determined on input. Conforming applications  
 99231 shall not specify a *blocksize* value larger than 32 256. Default blocking when  
 99232 creating archives depends on the archive format. (See the **-x** option below.)
- 99233 **-c** Match all file or archive members except those specified by the *pattern* or *file*  
 99234 operands.

## pax

## Utilities

99235	-d	Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.
99236		
99237		
99238	-f <i>archive</i>	Specify the pathname of the input or output archive, overriding the default standard input (in <b>list</b> or <b>read</b> modes) or standard output ( <b>write</b> mode).
99239		
99240	-H	If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
99241		
99242		
99243		
99244		
99245		
99246		
99247	-i	Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file <b>/dev/tty</b> . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from <b>/dev/tty</b> . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if <b>/dev/tty</b> cannot be opened for reading and writing.
99248		
99249		
99250		
99251		
99252		
99253		
99254		
99255		
99256		
99257	-k	The results of extracting a hard link to a file that has been renamed during extraction are unspecified.
99258		
99259	-l	(The letter ell.) In <b>copy</b> mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with -H or -L, when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither -H nor -L is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.
99260		
99261		
99262		
99263		
99264		
99265		
99266		
99267	-L	If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
99268		
99269		
99270		
99271		
99272		
99273		
99274		
99275	-n	Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).
99276		
99277		
99278	-o <i>options</i>	Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more <comma>-separated keywords of the form:
99279		
99280		
99281		<i>keyword</i> [[:]= <i>value</i> ] [, <i>keyword</i> [[:]= <i>value</i> ], ...]

99282 Some keywords apply only to certain file formats, as indicated with each  
 99283 description. Use of keywords that are inapplicable to the file format being  
 99284 processed produces undefined results.

99285 Keywords in the *options* argument shall be a string that would be a valid portable  
 99286 filename as described in XBD Section 3.276 (on page 77).

99287 **Note:** Keywords are not expected to be filenames, merely to follow the same character  
 99288 composition rules as portable filenames.

99289 Keywords can be preceded with white space. The *value* field shall consist of zero or  
 99290 more characters; within *value*, the application shall precede any literal <comma>  
 99291 with a <backslash>, which shall be ignored, but preserves the <comma> as part of  
 99292 *value*. A <comma> as the final character, or a <comma> followed solely by white  
 99293 space as the final characters, in *options* shall be ignored. Multiple **-o** options can be  
 99294 specified; if keywords given to these multiple **-o** options conflict, the keywords  
 99295 and values appearing later in command line sequence shall take precedence and  
 99296 the earlier shall be silently ignored. The following keyword values of *options* shall  
 99297 be supported for the file formats as indicated:

99298 **delete=pattern**

99299 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*  
 99300 shall omit from extended header records that it produces any keywords  
 99301 matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore  
 99302 any keywords matching the string pattern in the extended header records. In  
 99303 both cases, matching shall be performed using the pattern matching notation  
 99304 described in Section 2.13.1 (on page 2332) and Section 2.13.2 (on page 2332).  
 99305 For example:

99306 **-o delete=security.\***

99307 would suppress security-related information. See *pax Extended Header* (on  
 99308 page 3019) for extended header record keyword usage.

99309 When multiple **-o delete=pattern** options are specified, the patterns shall be  
 99310 additive; all keywords matching the specified string patterns shall be omitted  
 99311 from extended header records that *pax* produces.

99312 **exthdr.name=string**

99313 (Applicable only to the **-x pax** format.) This keyword allows user control over  
 99314 the name that is written into the **ustar** header blocks for the extended header  
 99315 produced under the circumstances described in *pax Header Block* (on page  
 99316 3019). The name shall be the contents of *string*, after the following character  
 99317 substitutions have been made:

<i>string</i> Includes:	Replaced by:
%d	The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname.
%f	The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.
%p	The process ID of the <i>pax</i> process.
%%	A ' % ' character.

99326 Any other ' % ' characters in *string* produce undefined results.

99327 If no **-o exthdr.name=string** is specified, *pax* shall use the following default

99328 value:  
 99329 %d/PaxHeaders.%p/%f

**globexthdr.name=string**

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records with **ustar** header blocks that will be treated as regular files by previous versions of *pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of *string*, after the following character substitutions have been made:

<i>string</i> Includes:	Replaced by:
%n	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
%p	The process ID of the <i>pax</i> process.
%%	A ' % ' character.

Any other ' % ' characters in *string* produce undefined results.

If no **-o globexthdr.name=string** is specified, *pax* shall use the following default value:

\$TMPDIR/GlobalHead.%p.%n

where \$TMPDIR represents the value of the *TMPDIR* environment variable. If *TMPDIR* is not set, *pax* shall use **/tmp**.

**invalid=action**

(Applicable only to the **-x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

- In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)
- In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).
- In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

The following mutually-exclusive values of the *action* argument are supported:

**binary** In **write** mode, *pax* shall generate a **hdrcharset=BINARY** extended header record for each file with a filename, link name, group name, owner name, or any other field in an extended header record that cannot be translated to the UTF-8 codeset, allowing the archive to contain the files with unencoded extended header record values. In **read** or **copy** mode, *pax* shall

- 99373 use the values specified in the header without translation,  
 99374 regardless of whether this may overwrite an existing file with a  
 99375 valid name. In **list** mode, *pax* shall behave identically to the  
 99376 **bypass** action.
- 99377 **bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no  
 99378 change to the destination hierarchy. In **list** mode, *pax* shall write  
 99379 all requested valid values for the file, but its method for writing  
 99380 invalid values is unspecified.
- 99381 **rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in  
 99382 effect for each file with invalid filename or link name values,  
 99383 allowing the user to provide a replacement name interactively.  
 99384 In **list** mode, *pax* shall behave identically to the **bypass** action.
- 99385 **UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link  
 99386 name, owner name, or any other field in an extended header  
 99387 record cannot be translated from the **pax** UTF-8 codeset format  
 99388 to the codeset and current locale of the implementation, *pax* shall  
 99389 use the actual UTF-8 encoding for the name. If a **hdrcharset**  
 99390 extended header record is in effect for this file, the character set  
 99391 specified by that record shall be used instead of UTF-8. If a  
 99392 **hdrcharset=BINARY** extended header record is in effect for this  
 99393 file, no translation shall be performed.
- 99394 **write** In **read** or **copy** mode, *pax* shall write the file, translating the  
 99395 name, regardless of whether this may overwrite an existing file  
 99396 with a valid name. In **list** mode, *pax* shall behave identically to  
 99397 the **bypass** action.
- 99398 If no **-o invalid=option** is specified, *pax* shall act as if **-o invalid=bypass** were  
 99399 specified. Any overwriting of existing files that may be allowed by the  
 99400 **-o invalid=** actions shall be subject to permission (**-p**) and modification time  
 99401 (**-u**) restrictions, and shall be suppressed if the **-k** option is also specified.
- 99402 **linkdata**  
 99403 (Applicable only to the **-x pax** format.) In **write** mode, *pax* shall write the  
 99404 contents of a file to the archive even when that file is merely a hard link to a  
 99405 file whose contents have already been written to the archive.
- 99406 **listopt=format**  
 99407 This keyword specifies the output format of the table of contents produced  
 99408 when the **-v** option is specified in **list** mode. See [List Mode Format](#)  
 99409 [Specifications](#) (on page 3014). To avoid ambiguity, the **listopt=format** shall be  
 99410 the only or final **keyword=value** pair in a **-o** option-argument; all characters  
 99411 in the remainder of the option-argument shall be considered part of the format  
 99412 string. When multiple **-olistopt=format** options are specified, the format  
 99413 strings shall be considered a single, concatenated string, evaluated in  
 99414 command line order.
- 99415 **times**  
 99416 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*  
 99417 shall include **atime** and **mtime** extended header records for each file. See [pax](#)  
 99418 [Extended Header File Times](#) (on page 3023).
- 99419 In addition to these keywords, if the **-x pax** format is specified, any of the

99420 keywords and values defined in [pax Extended Header](#) (on page 3019), including  
 99421 implementation extensions, can be used in `-o` option-arguments, in either of two  
 99422 modes:

99423 **keyword=***value*

99424 When used in **write** or **copy** mode, these keyword/value pairs shall be  
 99425 included at the beginning of the archive as **typeflag g** global extended header  
 99426 records. When used in **read** or **list** mode, these keyword/value pairs shall act  
 99427 as if they had been at the beginning of the archive as **typeflag g** global  
 99428 extended header records.

99429 **keyword:=***value*

99430 When used in **write** or **copy** mode, these keyword/value pairs shall be  
 99431 included as records at the beginning of a **typeflag x** extended header for each  
 99432 file. (This shall be equivalent to the `<equals-sign>` form except that it creates  
 99433 no **typeflag g** global extended header records.) When used in **read** or **list**  
 99434 mode, these keyword/value pairs shall act as if they were included as records  
 99435 at the end of each extended header; thus, they shall override any global or file-  
 99436 specific extended header record keywords of the same names. For example, in  
 99437 the command:

```
99438 pax -r -o "  
99439 gname:=mygroup,  
99440 " <archive
```

99441 the group name will be forced to a new value for all files read from the  
 99442 archive.

99443 The precedence of `-o` keywords over various fields in the archive is described in  
 99444 [pax Extended Header Keyword Precedence](#) (on page 3022).

99445 **-p** *string* Specify one or more file characteristic options (privileges). The *string* option-  
 99446 argument shall be a string specifying file characteristics to be retained or discarded  
 99447 on extraction. The string shall consist of the specification characters *a*, *e*, *m*, *o*, and  
 99448 *p*. Other implementation-defined characters can be included. Multiple  
 99449 characteristics can be concatenated within the same string and multiple `-p` options  
 99450 can be specified. The meaning of the specification characters are as follows:

99451 *a* Do not preserve file access times.

99452 *e* Preserve the user ID, group ID, file mode bits (see [XBD Section 3.169](#), on page  
 99453 60), access time, modification time, and any other implementation-defined file  
 99454 characteristics.

99455 *m* Do not preserve file modification times.

99456 *o* Preserve the user ID and group ID.

99457 *p* Preserve the file mode bits. Other implementation-defined file mode attributes  
 99458 may be preserved.

99459 In the preceding list, “preserve” indicates that an attribute stored in the archive  
 99460 shall be given to the extracted file, subject to the permissions of the invoking  
 99461 process. The access and modification times of the file shall be preserved unless  
 99462 otherwise specified with the `-p` option or not stored in the archive. All attributes  
 99463 that are not preserved shall be determined as part of the normal file creation action  
 99464 (see [Section 1.1.1.4](#), on page 2280).

- 99465 If neither the `e` nor the `o` specification character is specified, or the user ID and  
 99466 group ID are not preserved for any reason, *pax* shall not set the `S_ISUID` and  
 99467 `S_ISGID` bits of the file mode.
- 99468 If the preservation of any of these items fails for any reason, *pax* shall write a  
 99469 diagnostic message to standard error. Failure to preserve these items shall affect  
 99470 the final exit status, but shall not cause the extracted file to be deleted.
- 99471 If file characteristic letters in any of the *string* option-arguments are duplicated or  
 99472 conflict with each other, the ones given last shall take precedence. For example, if  
 99473 `-p_eme` is specified, file modification times are preserved.
- 99474 `-s replstr` Modify file or archive member names named by *pattern* or *file* operands according  
 99475 to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts  
 99476 of “address” and “line” are meaningless in the context of the *pax* utility, and shall  
 99477 not be supplied. The format shall be:
- 99478 `-s /old/new/ [gp]`
- 99479 where as in *ed*, *old* is a basic regular expression and *new* can contain an  
 99480 `<ampersand>`, `'\n'` (where *n* is a digit) back-references, or subexpression  
 99481 matching. The *old* string shall also be permitted to contain `<newline>` characters.
- 99482 Any non-null character can be used as a delimiter (`'/'` shown here). Multiple `-s`  
 99483 expressions can be specified; the expressions shall be applied in the order  
 99484 specified, terminating with the first successful substitution. The optional trailing  
 99485 `'g'` is as defined in the *ed* utility. The optional trailing `'p'` shall cause successful  
 99486 substitutions to be written to standard error. File or archive member names that  
 99487 substitute to the empty string shall be ignored when reading and writing archives.
- 99488 `-t` When reading files from the file system, and if the user has the permissions  
 99489 required by *utime()* to do so, set the access time of each file read to the access time  
 99490 that it had before being read by *pax*.
- 99491 `-u` Ignore files that are older (having a less recent file modification time) than a pre-  
 99492 existing file or archive member with the same name. In **read** mode, an archive  
 99493 member with the same name as a file in the file system shall be extracted if the  
 99494 archive member is newer than the file. In **write** mode, an archive file member with  
 99495 the same name as a file in the file system shall be superseded if the file is newer  
 99496 than the archive member. If `-a` is also specified, this is accomplished by appending  
 99497 to the archive; otherwise, it is unspecified whether this is accomplished by actual  
 99498 replacement in the archive or by appending to the archive. In **copy** mode, the file  
 99499 in the destination hierarchy shall be replaced by the file in the source hierarchy or  
 99500 by a link to the file in the source hierarchy if the file in the source hierarchy is  
 99501 newer.
- 99502 `-v` In **list** mode, produce a verbose table of contents (see the `STDOUT` section).  
 99503 Otherwise, write archive member pathnames to standard error (see the `STDERR`  
 99504 section).
- 99505 `-x format` Specify the output archive format. The *pax* utility shall support the following  
 99506 formats:
- 99507 **cpio** The **cpio** interchange format; see the EXTENDED DESCRIPTION  
 99508 section. The default *blocksize* for this format for character special  
 99509 archive files shall be 5120. Implementations shall support all  
 99510 *blocksize* values less than or equal to 32 256 that are multiples of 512.

- 99511            **pax**            The **pax** interchange format; see the EXTENDED DESCRIPTION  
99512 section. The default *blocksize* for this format for character special  
99513 archive files shall be 5120. Implementations shall support all  
99514 *blocksize* values less than or equal to 32 256 that are multiples of 512.
- 99515            **ustar**            The **tar** interchange format; see the EXTENDED DESCRIPTION  
99516 section. The default *blocksize* for this format for character special  
99517 archive files shall be 10 240. Implementations shall support all  
99518 *blocksize* values less than or equal to 32 256 that are multiples of 512.
- 99519            Implementation-defined formats shall specify a default block size as well as any  
99520 other block sizes supported for character special archive files.
- 99521            Any attempt to append to an archive file in a format different from the existing  
99522 archive format shall cause *pax* to exit immediately with a non-zero exit status.
- 99523            **-X**            When traversing the file hierarchy specified by a pathname, *pax* shall not descend  
99524 into directories that have a different device ID (*st\_dev*; see the System Interfaces  
99525 volume of POSIX.1-2008, *stat()*).
- 99526            Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
99527 an error and the last option specified shall determine the behavior of the utility.
- 99528            The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)  
99529 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-  
99530 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i**  
99531 options shall modify, in that order, the names of the selected files. The **-v** option shall write  
99532 names resulting from these modifications.
- 99533            In **write** mode, the files shall be selected based on the user-specified pathnames as modified by  
99534 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of  
99535 these selected files. The **-v** option shall write names resulting from these modifications.
- 99536            If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is  
99537 newer than the file to which it is compared.
- 99538            **List Mode Format Specifications**
- 99539            In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each  
99540 selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.  
99541 The *format* argument shall be used as the *format* string described in XBD Chapter 5 (on page 121),  
99542 with the exceptions 1. through 5. defined in the EXTENDED DESCRIPTION section of *printf*,  
99543 plus the following exceptions:
- 99544            6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion  
99545 argument is defined by the value of *keyword*. The implementation shall support the  
99546 following keywords:
- 99547            — Any of the Field Name entries in Table 4-14 (on page 3024) and Table 4-16 (on page  
99548 3027). The implementation may support the *cpio* keywords without the leading *c\_* in  
99549 addition to the form required by Table 4-16 (on page 3027).
- 99550            — Any keyword defined for the extended header in **pax Extended Header** (on page  
99551 3019).
- 99552            — Any keyword provided as an implementation-defined extension within the extended  
99553 header defined in **pax Extended Header** (on page 3019).
- 99554            For example, the sequence "%(charset)s" is the string value of the name of the character

- 99555 set in the extended header.
- 99556 The result of the keyword conversion argument shall be the value from the applicable  
99557 header field or extended header, without any trailing NULs.
- 99558 All keyword values used as conversion arguments shall be translated from the UTF-8  
99559 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to  
99560 the character set appropriate for the local file system, user database, and so on, as  
99561 applicable.
- 99562 7. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T**  
99563 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where  
99564 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**  
99565 and the default subformat shall be:
- 99566 %b %e %H:%M %Y
- 99567 8. An additional conversion specifier character, **M**, shall be used to specify the file mode string  
99568 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.  
99569 For example, `%.1M` writes the single character corresponding to the *<entry type>* field of the  
99570 *ls -l* command.
- 99571 9. An additional conversion specifier character, **D**, shall be used to specify the device for block  
99572 or special files, if applicable, in an implementation-defined format. If not applicable, and  
99573 (*keyword*) is specified, then this conversion shall be equivalent to `%(keyword)u`. If not  
99574 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to `<space>`.
- 99575 10. An additional conversion specifier character, **F**, shall be used to specify a pathname. The **F**  
99576 conversion character can be preceded by a sequence of `<comma>`-separated keywords:
- 99577 (*keyword*[, *keyword*] ... )
- 99578 The values for all the keywords that are non-null shall be concatenated together, each  
99579 separated by a `'/'`. The default shall be (**path**) if the keyword **path** is defined; otherwise,  
99580 the default shall be (**prefix,name**).
- 99581 11. An additional conversion specifier character, **L**, shall be used to specify a symbolic link  
99582 expansion. If the current file is a symbolic link, then `%L` shall expand to:
- 99583 "%s -> %s", *<value of keyword>*, *<contents of link>*
- 99584 Otherwise, the `%L` conversion specification shall be the equivalent of `%F`.

**OPERANDS**

99585 The following operands shall be supported:

- 99586 *directory* The destination directory pathname for **copy** mode.
- 99587 *file* A pathname of a file to be copied or archived.
- 99588 *pattern* A pattern matching one or more pathnames of archive members. A pattern must  
99589 be given in the name-generating notation of the pattern matching notation in  
99590 [Section 2.13](#) (on page 2332), including the filename expansion rules in [Section](#)  
99591 [2.13.3](#) (on page 2333). The default, if no *pattern* is specified, is to select all members  
99592 in the archive.

**STDIN**

99594 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a  
99595 text file containing a list of pathnames, one per line, without leading or trailing `<blank>`  
99596 characters.

99598 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.  
 99599 Otherwise, the standard input shall not be used.

#### 99600 INPUT FILES

99601 The input file named by the *archive* option-argument, or standard input when the archive is read  
 99602 from there, shall be a file formatted according to one of the specifications in the EXTENDED  
 99603 DESCRIPTION section or some other implementation-defined format.  
 99604 The file **/dev/tty** shall be used to write prompts and read responses.

#### 99605 ENVIRONMENT VARIABLES

99606 The following environment variables shall affect the execution of *pax*:

99607 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 99608 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
 99609 used to determine the values of locale categories.)

99610 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 99611 internationalization variables.

#### 99612 LC\_COLLATE

99613 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 99614 character collating elements used in the pattern matching expressions for the  
 99615 *pattern* operand, the basic regular expression for the **-s** option, and the extended  
 99616 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
 99617 category.

99618 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 99619 characters (for example, single-byte as opposed to multi-byte characters in  
 99620 arguments and input files) and the behavior of character classes used in the extended  
 99621 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
 99622 category, and pattern matching.

#### 99623 LC\_MESSAGES

99624 Determine the locale used to process affirmative responses, and the locale used to  
 99625 affect the format and contents of diagnostic messages and prompts written to  
 99626 standard error.

99627 **LC\_TIME** Determine the format and contents of date and time strings when the **-v** option is  
 99628 specified.

99629 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

99630 **TMPDIR** Determine the pathname that provides part of the default global extended header  
 99631 record file, as described for the **-o globexthdr=** keyword in the OPTIONS section.

99632 **TZ** Determine the timezone used to calculate date and time strings when the **-v** option  
 99633 is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

#### 99634 ASYNCHRONOUS EVENTS

99635 Default.

#### 99636 STDOUT

99637 In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted  
 99638 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
 99639 implementation-defined format (see **-x format**).

99640 In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall  
 99641 be written to standard output using the format described under [List Mode Format Specifications](#)

99642 (on page 3014). In **list** mode without the **-olistopt=format** option, the table of contents of the  
 99643 selected archive members shall be written to standard output using the following format:

99644 "%s\n", <pathname>

99645 If the **-v** option is specified in **list** mode, the table of contents of the selected archive members  
 99646 shall be written to standard output using the following formats.

99647 For pathnames representing hard links to previous members of the archive:

99648 "%sΔ==Δ%s\n", <ls -l listing>, <linkname>

99649 For all other pathnames:

99650 "%s\n", <ls -l listing>

99651 where <ls -l listing> shall be the format specified by the *ls* utility with the **-l** option. When  
 99652 writing pathnames in this format, it is unspecified what is written for fields for which the  
 99653 underlying archive format does not have the correct information, although the correct number of  
 99654 <blank>-separated fields shall be written.

99655 In **list** mode, standard output shall not be buffered more than a line at a time.

#### 99656 STDERR

99657 If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the  
 99658 standard error output using the following format:

99659 "%s\n", <pathname>

99660 These pathnames shall be written as soon as processing is begun on the file or archive member,  
 99661 and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is  
 99662 written when the file has been read or written.

99663 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be  
 99664 written to standard error in the following format:

99665 "%sΔ>>Δ%s\n", <original pathname>, <new pathname>

99666 In all operating modes of *pax*, optional messages of unspecified format concerning the input  
 99667 archive format and volume number, the number of files, blocks, volumes, and media parts as  
 99668 well as other diagnostic messages may be written to standard error.

99669 In all formats, for both standard output and standard error, it is unspecified how non-printable  
 99670 characters in pathnames or link names are written.

99671 When using the **-xpax** archive format, if a filename, link name, group name, owner name, or any  
 99672 other field in an extended header record cannot be translated between the codeset in use for that  
 99673 extended header record and the character set of the current locale, *pax* shall write a diagnostic  
 99674 message to standard error, shall process the file as described for the **-o invalid=** option, and then  
 99675 shall continue processing with the next file.

#### 99676 OUTPUT FILES

99677 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the  
 99678 copied output files shall be the type of the file being copied. In either mode, existing files in the  
 99679 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),  
 99680 and invalid-value (**-oinvalid=**) tests allow it.

99681 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted  
 99682 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
 99683 implementation-defined format.

99684 **EXTENDED DESCRIPTION**

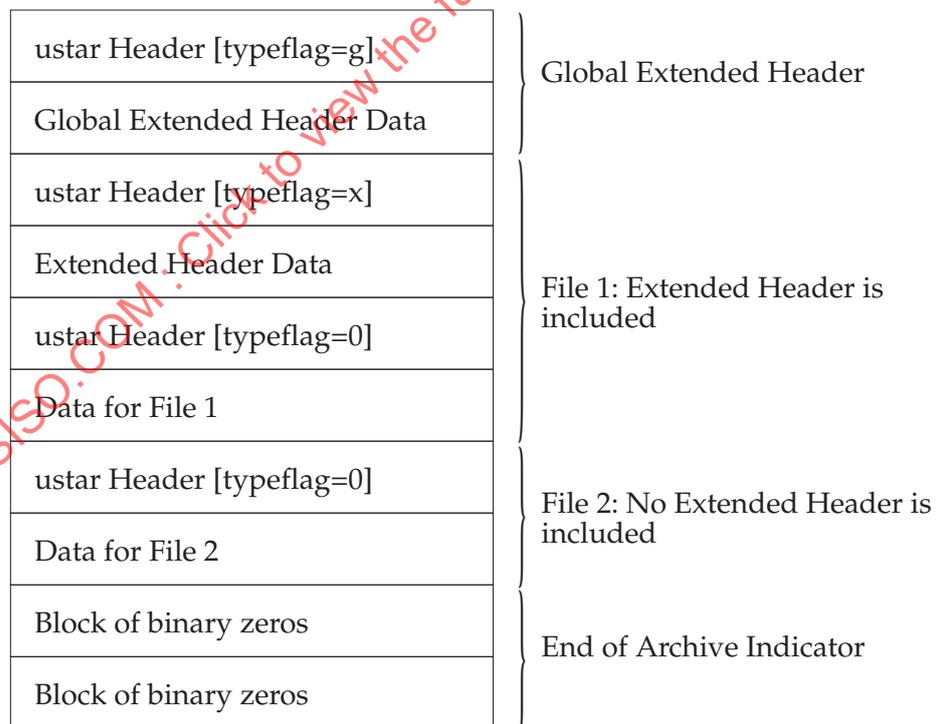
99685 **pax Interchange Format**

99686 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The  
 99687 physical layout of the archive shall be identical to the **ustar** format described in **ustar**  
 99688 **Interchange Format** (on page 3023). Each file archived shall be represented by the following  
 99689 sequence:

- 99690 • An optional header block with extended header records. This header block is of the form  
 99691 described in **pax Header Block** (on page 3019), with a *typeflag* value of **x** or **g**. The  
 99692 extended header records, described in **pax Extended Header** (on page 3019), shall be  
 99693 included as the data for this header block.
- 99694 • A header block that describes the file. Any fields in the preceding optional extended  
 99695 header shall override the associated fields in this header block for this file.
- 99696 • Zero or more blocks that contain the contents of the file.

99697 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,  
 99698 interpreted as an end-of-archive indicator.

99699 A schematic of an example archive with global extended header records and two actual files is  
 99700 shown in **Figure 4-1**. In the example, the second file in the archive has no extended header  
 99701 preceding it, presumably because it has no need for extended attributes.



**Figure 4-1** pax Format Archive Example

99702

99703 **pax Header Block**

99704 The **pax** header block shall be identical to the **ustar** header block described in **ustar Interchange**  
 99705 **Format** (on page 3023), except that two additional *typeflag* values are defined:

- 99706 x Represents extended header records for the following file in the archive (which shall have  
 99707 its own **ustar** header block). The format of these extended header records shall be as  
 99708 described in **pax Extended Header**.
- 99709 g Represents global extended header records for the following files in the archive. The format  
 99710 of these extended header records shall be as described in **pax Extended Header**. Each value  
 99711 shall affect all subsequent files that do not override that value in their own extended header  
 99712 record and until another global extended header record is reached that provides another  
 99713 value for the same field. The *typeflag* **g** global headers should not be used with interchange  
 99714 media that could suffer partial data loss in transporting the archive.

99715 For both of these types, the *size* field shall be the size of the extended header records in octets.  
 99716 The other fields in the header block are not meaningful to this version of the *pax* utility.  
 99717 However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2:1993 standard,  
 99718 the header block fields are used to create a regular file that contains the extended header records  
 99719 as data. Therefore, header block field values should be selected to provide reasonable file access  
 99720 to this regular file.

99721 A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit  
 99722 one) (hard link) may be included, which means that the size field may be greater than zero.  
 99723 Archives created by *pax -o linkdata* shall include these data blocks with the hard links.

99724 **pax Extended Header**

99725 A **pax** extended header contains values that are inappropriate for the **ustar** header block because  
 99726 of limitations in that format: fields requiring a character encoding other than that described in  
 99727 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**  
 99728 header, and fields whose format or length do not fit the requirements of the **ustar** header. The  
 99729 values in an extended header add attributes to the following file (or files; see the description of  
 99730 the *typeflag* **g** header block) or override values in the following header block(s), as indicated in  
 99731 the following list of keywords.

99732 An extended header shall consist of one or more records, each constructed as follows:

99733 "%d %s=%s\n" <length>, <keyword>, <value>

99734 The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard  
 99735 UTF-8 encoding. The <length> field, <blank>, <equals-sign>, and <newline> shown shall be  
 99736 limited to the portable character set, as encoded in UTF-8. The <keyword> fields can be any  
 99737 UTF-8 characters. The <length> field shall be the decimal length of the extended header record  
 99738 in octets, including the trailing <newline>. If there is a **hdrcharset** extended header in effect for  
 99739 a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be  
 99740 encoded using the character set specified by the **hdrcharset** extended header record; otherwise,  
 99741 the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by  
 99742 POSIX.1-2008 shall be encoded using UTF-8.

99743 The <keyword> field shall be one of the entries from the following list or a keyword provided as  
 99744 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and  
 99745 periods are reserved for future standardization. A keyword shall not include an <equals-sign>.  
 99746 (In the following list, the notations "file(s)" or "block(s)" is used to acknowledge that a keyword  
 99747 affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after  
 99748 *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode

**pax**

99749 shall apply only when such a record has not already been provided through the use of the **-o**  
 99750 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with  
 99751 applicable extended header records and then extracted.)

99752 **atime** The file access time for the following file(s), equivalent to the value of the *st\_atime*  
 99753 member of the **stat** structure for a file, as described by the *stat()* function. The  
 99754 access time shall be restored if the process has appropriate privileges required to  
 99755 do so. The format of the *<value>* shall be as described in *pax Extended Header File*  
 99756 *Times* (on page 3023).

99757 **charset** The name of the character set used to encode the data in the following file(s). The  
 99758 entries in the following table are defined to refer to known standards; additional  
 99759 names may be agreed on between the originator and recipient.

<i>&lt;value&gt;</i>	Formal Standard
ISO-IRΔ646Δ1990	ISO/IEC 646: 1990
ISO-IRΔ8859Δ1Δ1998	ISO/IEC 8859-1: 1998
ISO-IRΔ8859Δ2Δ1999	ISO/IEC 8859-2: 1999
ISO-IRΔ8859Δ3Δ1999	ISO/IEC 8859-3: 1999
ISO-IRΔ8859Δ4Δ1998	ISO/IEC 8859-4: 1998
ISO-IRΔ8859Δ5Δ1999	ISO/IEC 8859-5: 1999
ISO-IRΔ8859Δ6Δ1999	ISO/IEC 8859-6: 1999
ISO-IRΔ8859Δ7Δ1987	ISO/IEC 8859-7: 1987
ISO-IRΔ8859Δ8Δ1999	ISO/IEC 8859-8: 1999
ISO-IRΔ8859Δ9Δ1999	ISO/IEC 8859-9: 1999
ISO-IRΔ8859Δ10Δ1998	ISO/IEC 8859-10: 1998
ISO-IRΔ8859Δ13Δ1998	ISO/IEC 8859-13: 1998
ISO-IRΔ8859Δ14Δ1998	ISO/IEC 8859-14: 1998
ISO-IRΔ8859Δ15Δ1999	ISO/IEC 8859-15: 1999
ISO-IRΔ10646Δ2000	ISO/IEC 10646: 2000
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

99778 The encoding is included in an extended header for information only; when *pax* is  
 99779 used as described in POSIX.1-2008, it shall not translate the file data into any other  
 99780 encoding. The **BINARY** entry indicates unencoded binary data.

99781 When used in **write** or **copy** mode, it is implementation-defined whether *pax*  
 99782 includes a **charset** extended header record for a file.

99783 **comment** A series of characters used as a comment. All characters in the *<value>* field shall  
 99784 be ignored by *pax*.

99785 **gid** The group ID of the group that owns the file, expressed as a decimal number using  
 99786 digits from the ISO/IEC 646: 1991 standard. This record shall override the *gid* field  
 99787 in the following header block(s). When used in **write** or **copy** mode, *pax* shall  
 99788 include a *gid* extended header record for each file whose group ID is greater than  
 99789 2 097 151 (octal 7777 777).

99790 **gname** The group of the file(s), formatted as a group name in the group database. This  
 99791 record shall override the *gid* and *gname* fields in the following header block(s), and  
 99792 any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall  
 99793 translate the name from the encoding in the header record to the character set  
 99794 appropriate for the group database on the receiving system. If any of the characters  
 99795 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the

99796 –**oinvalid=binary** option is specified, the results are implementation-defined.  
 99797 When used in **write** or **copy** mode, *pax* shall include a **gname** extended header  
 99798 record for each file whose group name cannot be represented entirely with the  
 99799 letters and digits of the portable character set.

99800 **hdrcharset** The name of the character set used to encode the value field of the **gname**,  
 99801 **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the  
 99802 following table are defined to refer to known standards; additional names may be  
 99803 agreed between the originator and the recipient.

	<value>	Formal Standard
99804	ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
99805	BINARY	None.
99806		

99807 If no **hdrcharset** extended header record is specified, the default character set used  
 99808 to encode all values in extended header records shall be the ISO/IEC 10646-1:2000  
 99809 standard UTF-8 encoding.

99810 The **BINARY** entry indicates that all values recorded in extended headers for  
 99811 affected files are unencoded binary data from the underlying system.

99812 **linkpath** The pathname of a link being created to another file, of any type, previously  
 99813 archived. This record shall override the *linkname* field in the following **ustar** header  
 99814 block(s). The following **ustar** header block shall determine the type of link created.  
 99815 If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it  
 99816 shall be a symbolic link and the **linkpath** value shall be the contents of the  
 99817 symbolic link. The *pax* utility shall translate the name of the link (contents of the  
 99818 symbolic link) from the encoding in the header to the character set appropriate for  
 99819 the local file system. When used in **write** or **copy** mode, *pax* shall include a  
 99820 **linkpath** extended header record for each link whose pathname cannot be  
 99821 represented entirely with the members of the portable character set other than  
 99822 NUL.

99823 **mtime** The file modification time of the following file(s), equivalent to the value of the  
 99824 *st\_mtime* member of the **stat** structure for a file, as described in the *stat()* function.  
 99825 This record shall override the *mtime* field in the following header block(s). The  
 99826 modification time shall be restored if the process has appropriate privileges  
 99827 required to do so. The format of the <value> shall be as described in [pax Extended](#)  
 99828 [Header File Times](#) (on page 3023).

99829 **path** The pathname of the following file(s). This record shall override the *name* and  
 99830 *prefix* fields in the following header block(s). The *pax* utility shall translate the  
 99831 pathname of the file from the encoding in the header to the character set  
 99832 appropriate for the local file system.

99833 When used in **write** or **copy** mode, *pax* shall include a *path* extended header record  
 99834 for each file whose pathname cannot be represented entirely with the members of  
 99835 the portable character set other than NUL.

99836 **realtime.any** The keywords prefixed by “realtime.” are reserved for future standardization.

99837 **security.any** The keywords prefixed by “security.” are reserved for future standardization.

99838 **size** The size of the file in octets, expressed as a decimal number using digits from the  
 99839 ISO/IEC 646:1991 standard. This record shall override the *size* field in the  
 99840 following header block(s). When used in **write** or **copy** mode, *pax* shall include a  
 99841 *size* extended header record for each file with a size value greater than 8 589 934 591

- 99842 (octal 77 777 777 777).
- 99843 **uid** The user ID of the file owner, expressed as a decimal number using digits from the  
99844 ISO/IEC 646:1991 standard. This record shall override the *uid* field in the  
99845 following header block(s). When used in **write** or **copy** mode, *pax* shall include a  
99846 *uid* extended header record for each file whose owner ID is greater than 2097151  
99847 (octal 7777777).
- 99848 **uname** The owner of the following file(s), formatted as a user name in the user database.  
99849 This record shall override the *uid* and *uname* fields in the following header block(s),  
99850 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*  
99851 shall translate the name from the encoding in the header record to the character set  
99852 appropriate for the user database on the receiving system. If any of the characters  
99853 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the  
99854 **-oinvalid=binary** option is specified, the results are implementation-defined.  
99855 When used in **write** or **copy** mode, *pax* shall include a **uname** extended header  
99856 record for each file whose user name cannot be represented entirely with the letters  
99857 and digits of the portable character set.
- 99858 If the *<value>* field is zero length, it shall delete any header block field, previously entered  
99859 extended header value, or global extended header value of the same name.
- 99860 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a  
99861 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block  
99862 field.
- 99863 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the  
99864 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**  
99865 header block fields in Table 4-14 (on page 3024) shall apply to the extended header records.
- 99866 **pax Extended Header Keyword Precedence**
- 99867 This section describes the precedence in which the various header records and fields and  
99868 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or  
99869 **list** modes, it shall determine a file attribute in the following sequence:
- 99870 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step  
99871 7., if applicable, or ignored otherwise.
  - 99872 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
  - 99873 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
  - 99874 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the  
99875 *<value>*. When extended header records conflict, the last one given in the header shall  
99876 take precedence.
  - 99877 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
  - 99878 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be  
99879 assigned the *<value>*. When global extended header records conflict, the last one given in  
99880 the global header shall take precedence.
  - 99881 7. Otherwise, the attribute shall be determined from the **ustar** header block.

99882 **pax Extended Header File Times**

99883 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's  
 99884 modification time cannot be represented exactly in the **ustar** header logical record described in  
 99885 **ustar Interchange Format**. This can occur if the time is out of **ustar** range, or if the file system of  
 99886 the underlying implementation supports non-integer time granularities and the time is not an  
 99887 integer. All of these time records shall be formatted as a decimal representation of the time in  
 99888 seconds since the Epoch. If a <period> ( ' . ' ) decimal point character is present, the digits to the  
 99889 right of the point shall represent the units of a subsecond timing granularity, where the first digit  
 99890 is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy**  
 99891 mode, the *pax* utility shall truncate the time of a file to the greatest value that is not greater than  
 99892 the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it  
 99893 can be represented exactly as a decimal number, and otherwise shall generate only enough digits  
 99894 so that the same time shall be recovered if the file is extracted on a system whose underlying  
 99895 implementation supports the same time granularity.

99896 **ustar Interchange Format**

99897 A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a  
 99898 fixed-size logical record of 512 octets (see below). Although this format may be thought of as  
 99899 being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of  
 99900 transportable media are not excluded. Each file archived shall be represented by a header logical  
 99901 record that describes the file, followed by zero or more logical records that give the contents of  
 99902 the file. At the end of the archive file there shall be two 512-octet logical records filled with  
 99903 binary zeros, interpreted as an end-of-archive indicator.

99904 The logical records may be grouped for physical I/O operations, as described under the  
 99905 **-b** *blocksize* and **-x** **ustar** options. Each group of logical records may be written with a single  
 99906 operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a  
 99907 single tape physical block. The last physical block shall always be the full size, so logical records  
 99908 after the two zero logical records may contain undefined data.

99909 The header logical record shall be structured as shown in the following table. All lengths and  
 99910 offsets are in decimal.

99911

Table 4-14 ustar Header Block

99912

99913

99914

99915

99916

99917

99918

99919

99920

99921

99922

99923

99924

99925

99926

99927

99928

Field Name	Octet Offset	Length (in Octets)
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

99929

99930

99931

99932

99933

99934

99935

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646:1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of <slash> and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

99936

99937

99938

99939

99940

However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described in POSIX.1-2008. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

99941

99942

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

99943

99944

99945

99946

99947

99948

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

99949

99950

99951

99952

99953

99954

99955

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a <slash> character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any part of the file—header or data—on the medium.

99956

99957

The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall

99958 notify the user of the error, and shall not attempt to store the link on the medium.

99959 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit  
99960 representation. The encoded bits shall represent the following values:

99961 **Table 4-15** *ustar mode* Field

Bit Value	POSIX.1-2008 Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

99975 When appropriate privileges are required to set one of these mode bits, and the user restoring  
99976 the files from the archive does not have appropriate privileges, the mode bits for which the user  
99977 does not have appropriate privileges shall be ignored. Some of the mode bits in the archive  
99978 format are not mentioned elsewhere in this volume of POSIX.1-2008. If the implementation does  
99979 not support those bits, they may be ignored.

99980 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

99981 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type  
99982 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to  
99983 specify a file of type 5 (directory), the *size* field shall be interpreted as described under the  
99984 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*  
99985 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*  
99986 field is unspecified by this volume of POSIX.1-2008, and no data logical records shall be stored  
99987 on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the  
99988 *typeflag* field is set to any other value, the number of logical records written following the header  
99989 shall be  $(size+511)/512$ , ignoring any fraction in the result of the division.

99990 The *mtime* field shall be the modification time of the file at the time it was archived. It is the  
99991 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained  
99992 from the *stat()* function.

99993 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of  
99994 the simple sum of all octets in the header logical record. Each octet in the header shall be treated  
99995 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the  
99996 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is  
99997 treated as if it were all <space> characters.

99998 The *typeflag* field specifies the type of file archived. If a particular implementation does not  
99999 recognize the type, or the user does not have appropriate privileges to create that type, the file  
100000 shall be extracted as if it were a regular file if the file type is defined to have a meaning for the  
100001 *size* field that could cause data logical records to be written on the medium (see the previous  
100002 description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error  
100003 indicating that the conversion took place. All of the *typeflag* fields shall be coded in the

100004	ISO/IEC 646: 1991 standard IRV:
100005	0
100006	Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero (' \0 ') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646: 1991 standard IRV ' 0 '.
100007	
100008	
100009	1
100010	Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length.
100011	
100012	
100013	
100014	2
100015	Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field.
100016	3, 4
100017	Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-2008. Implementations may map the device specifications to their own local specification or may ignore the entry.
100018	
100019	
100020	5
100021	Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field.
100022	
100023	
100024	
100025	6
100026	Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.
100027	7
100028	Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).
100029	
100030	A–Z
100031	The letters ' A ' to ' Z ', inclusive, are reserved for custom implementations. All other values are reserved for future versions of this standard.
100032	It is unspecified whether files with pathnames that refer to the same directory entry are archived as linked files or as separate files. If they are archived as linked files, this means that attempting to extract both pathnames from the resulting archive will always cause an error (unless the <i>-u</i> option is used) because the link cannot be created.
100033	
100034	
100035	
100036	It is unspecified whether files with the same device and file serial numbers being appended to an archive are treated as linked files to members that were in the archive before the append.
100037	
100038	Attempts to archive a socket using <i>ustar</i> interchange format shall produce a diagnostic message. Handling of other file types is implementation-defined.
100039	
100040	The <i>magic</i> field is the specification that this archive was output in this archive format. If this field contains <i>ustar</i> (the five characters from the ISO/IEC 646: 1991 standard IRV shown followed by NUL), the <i>uname</i> and <i>gname</i> fields shall contain the ISO/IEC 646: 1991 standard IRV representation of the owner and group of the file, respectively (truncated to fit, if necessary).
100041	
100042	
100043	
100044	
100045	When the file is restored by a privileged, protection-preserving version of the utility, the user and group databases shall be scanned for these names. If found, the user and group IDs contained within these files shall be used rather than the values contained within the <i>uid</i> and <i>gid</i> fields.
100046	
100047	

100048 **cpio Interchange Format**

100049 The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that  
 100050 describes the file, the name of the file, and then the contents of the file.

100051 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used  
 100052 only to make physical I/O more efficient. The last group of blocks shall always be at the full  
 100053 size.

100054 For the octet-oriented **cpio** archive format, the individual entry information shall be in the order  
 100055 indicated and described by the following table; see also the **<cpio.h>** header.

100056 **Table 4-16** Octet-Oriented cpio Archive Entry

Header Field Name	Length (in Octets)	Interpreted as
<i>c_magic</i>	6	Octal number
<i>c_dev</i>	6	Octal number
<i>c_ino</i>	6	Octal number
<i>c_mode</i>	6	Octal number
<i>c_uid</i>	6	Octal number
<i>c_gid</i>	6	Octal number
<i>c_nlink</i>	6	Octal number
<i>c_rdev</i>	6	Octal number
<i>c_mtime</i>	11	Octal number
<i>c_namesize</i>	6	Octal number
<i>c_filesize</i>	11	Octal number
Filename Field Name	Length	Interpreted as
<i>c_name</i>	<i>c_namesize</i>	Pathname string
File Data Field Name	Length	Interpreted as
<i>c_filedata</i>	<i>c_filesize</i>	Data

100073 **cpio Header**

100074 For each file in the archive, a header as defined previously shall be written. The information in  
 100075 the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted  
 100076 as octal numbers. The octal numbers shall be extended to the necessary length by appending the  
 100077 ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result  
 100078 is written to the most-significant digit of the stream of octets first. The fields shall be interpreted  
 100079 as follows:

100080 *c\_magic* Identify the archive as being a transportable archive by containing the identifying  
 100081 value "070707".

100082 *c\_dev, c\_ino* Contains values that uniquely identify the file within the archive (that is, no files  
 100083 contain the same pair of *c\_dev* and *c\_ino* values unless they are links to the same  
 100084 file). The values shall be determined in an unspecified manner.

100085 *c\_mode* Contains the file type and access permissions as defined in the following table.

100086

Table 4-17 Values for cpio c\_mode Field

100087

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set <i>uid</i>
C_ISGID	002 000	Set <i>gid</i>
C_ISVTX	001 000	Reserved
File Type Name	Value	Indicates
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

100088

100089

100090

100091

100092

100093

100094

100095

100096

100097

100098

100099

100100

100101

100102

100103

100104

100105

100106

100107

100108

100109

100110

100111

100112

100113

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of POSIX.1-2008; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

100114

*c\_uid*

Contains the user ID of the owner.

100115

*c\_gid*

Contains the group ID of the group.

100116

*c\_nlink*

Contains a number greater than or equal to the number of links in the archive referencing the file. If the *-a* option is used to append to a *cpio* archive, then the *pax* utility need not account for the files in the existing part of the archive when calculating the *c\_nlink* values for the appended part of the archive, and need not alter the *c\_nlink* values in the existing part of the archive if additional files with the same *c\_dev* and *c\_ino* values are appended to the archive.

100117

100118

100119

100120

100121

100122

*c\_rdev*

Contains implementation-defined information for character or block special files.

100123

*c\_mtime*

Contains the latest time of modification of the file at the time the archive was created.

100124

100125

*c\_namesize*

Contains the length of the pathname, including the terminating NUL character.

100126

*c\_filesize*

Contains the length in octets of the data section following the header structure.

100127 **cpio Filename**

100128 The *c\_name* field shall contain the pathname of the file. The length of this field in octets is the  
100129 value of *c\_namesize*.

100130 If a filename is found on the medium that would create an invalid pathname, it is  
100131 implementation-defined whether the data from the file is stored on the file hierarchy and under  
100132 what name it is stored.

100133 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum  
100134 portability between implementations, names should be selected from characters represented by  
100135 the portable filename character set as octets with the most significant bit zero. If an  
100136 implementation supports the use of characters outside the portable filename character set in  
100137 names for files, users, and groups, one or more implementation-defined encodings of these  
100138 characters shall be provided for interchange purposes. However, the *pax* utility shall never create  
100139 filenames on the local system that cannot be accessed via the procedures described previously in  
100140 this volume of POSIX.1-2008. If a filename is found on the medium that would create an invalid  
100141 filename, it is implementation-defined whether the data from the file is stored on the local file  
100142 system and under what name it is stored. The *pax* utility may choose to ignore these files as long  
100143 as it produces an error indicating that the file is being ignored.

100144 **cpio File Data**

100145 Following *c\_name*, there shall be *c\_filesize* octets of data. Interpretation of such data occurs in a  
100146 manner dependent on the file. For regular files, the data shall consist of the contents of the file.  
100147 For symbolic links, the data shall consist of the contents of the symbolic link. If *c\_filesize* is zero,  
100148 no data shall be contained in *c\_filedata*.

100149 When restoring from an archive:

- 100150 • If the user does not have appropriate privileges to create a file of the specified type, *pax*  
100151 shall ignore the entry and write an error message to standard error.
- 100152 • Only regular files and symbolic links have data to be restored. Presuming a regular file  
100153 meets any selection criteria that might be imposed on the format-reading utility by the  
100154 user, such data shall be restored.
- 100155 • If a user does not have appropriate privileges to set a particular mode flag, the flag shall be  
100156 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this  
100157 volume of POSIX.1-2008. If the implementation does not support those flags, they may be  
100158 ignored.

100159 **cpio Special Entries**

100160 FIFO special files, directories, and the trailer shall be recorded with *c\_filesize* equal to zero.  
100161 Symbolic links shall be recorded with *c\_filesize* equal to the length of the contents of the symbolic  
100162 link. For other special files, *c\_filesize* is unspecified by this volume of POSIX.1-2008. The header  
100163 for the next file entry in the archive shall be written directly after the last octet of the file entry  
100164 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive;  
100165 the contents of octets in the last block of the archive following such a header are undefined.

100166 **EXIT STATUS**

100167 The following exit values shall be returned:

- 100168 0 All files were processed successfully.

100169 >0 An error occurred.

#### 100170 CONSEQUENCES OF ERRORS

100171 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an  
 100172 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a  
 100173 diagnostic message shall be written to standard error and a non-zero exit status shall be  
 100174 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*  
 100175 shall not, by default, create a second copy of the file.

100176 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may  
 100177 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a  
 100178 file of the same name as that specified by the user, but which is not the file the user wanted.  
 100179 Additionally, the file modes of extracted directories may have additional bits from the S\_IRWXU  
 100180 mask set as well as incorrect modification and access times.

#### 100181 APPLICATION USAGE

100182 Caution is advised when using the **-a** option to append to a *cpio* format archive. If any of the  
 100183 files being appended happen to be given the same *c\_dev* and *c\_ino* values as a file in the existing  
 100184 part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to  
 100185 use **-a** with *cpio* format except when it is done on the same system that the original archive was  
 100186 created on, and with the same *pax* utility, and in the knowledge that there has been little or no  
 100187 file system activity since the original archive was created that could lead to any of the files  
 100188 appended being given the same *c\_dev* and *c\_ino* values as an unrelated file in the existing part of  
 100189 the archive. Also, when (intentionally) appending additional links to a file in the existing part of  
 100190 the archive, the *c\_nlink* values in the modified archive can be smaller than the number of links to  
 100191 the file in the archive, which may mean that the links are not preserved on extraction.

100192 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*  
 100193 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**  
 100194 option also provides a consistent means of extending the ways in which future file attributes can  
 100195 be addressed, such as for enhanced security systems or high-performance files. Although it may  
 100196 seem complex, there are really two modes that are most commonly used:

100197 **-p e** "Preserve everything". This would be used by the historical superuser, someone with  
 100198 all appropriate privileges, to preserve all aspects of the files as they are recorded in the  
 100199 archive. The *e* flag is the sum of *o* and *p*, and other implementation-defined attributes.

100200 **-p p** "Preserve" the file mode bits. This would be used by the user with regular privileges  
 100201 who wished to preserve aspects of the file other than the ownership. The file times are  
 100202 preserved by default, but two other flags are offered to disable these and use the time  
 100203 of extraction.

100204 The one pathname per line format of standard input precludes pathnames containing <newline>  
 100205 characters. Although such pathnames violate the portable filename guidelines, they may exist  
 100206 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from  
 100207 historical archive programs. The problem can be avoided by listing filename arguments on the  
 100208 command line instead of on standard input.

100209 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this  
 100210 volume of POSIX.1-2008. Specifically, creating files of type block special or character special,  
 100211 restoring file access times unless the files are owned by the user (the **-t** option), or preserving file  
 100212 owner, group, and mode (the **-p** option) all probably require appropriate privileges.

100213 In **read** mode, implementations are permitted to overwrite files when the archive has multiple  
 100214 members with the same name. This may fail if permissions on the first version of the file do not  
 100215 permit it to be overwritten.

100216 The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ( $8 * 2^{30}$ ) in size.

100217 When archives containing binary header information are listed , the filenames printed may  
100218 cause strange behavior on some terminals.

#### 100219 EXAMPLES

100220 The following command:

```
100221 pax -w -f /dev/rmt/lm .
```

100222 copies the contents of the current directory to tape drive 1, medium density (assuming historical  
100223 System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

100224 The following commands:

```
100225 mkdir newdir
100226 pax -rw olddir newdir
```

100227 copy the *olddir* directory hierarchy to *newdir*.

```
100228 pax -r -s ',^//*usr//*,,' -f a.pax
```

100229 reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current  
100230 directory.

100231 Using the option:

```
100232 -o listopt="%M %(atime)T %(size)D %(name)s"
```

100233 overrides the default output description in Standard Output and instead writes:

```
100234 -rw-rw--- Jan 12 15:53 2003 1492 /usr/foo/bar
```

100235 Using the options:

```
100236 -o listopt='%L\t%(size)D\n%.7' \
100237 -o listopt='(name)s\n%(atime)T\n%T'
```

100238 overrides the default output description in Standard Output and instead writes:

```
100239 /usr/foo/bar -> /tmp 1492
100240 /usr/fo
100241 Jan 12 15:53 1991
100242 Jan 31 15:53 2003
```

#### 100243 RATIONALE

100244 The *pax* utility was new for the ISO POSIX-2: 1993 standard. It represents a peaceful compromise  
100245 between advocates of the historical *tar* and *cpio* utilities.

100246 A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio*  
100247 utility did not treat directories differently from other files, and to select a directory and its  
100248 contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory  
100249 matched every file in the file hierarchy it rooted.

100250 The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root.  
100251 The **-d** option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar*  
100252 **-style** behavior was chosen as the default because it was believed that this was the more  
100253 common usage and because *tar* is the more commonly available interface, as it was historically  
100254 provided on both System V and BSD implementations.

100255 The data interchange format specification in this volume of POSIX.1-2008 requires that processes  
100256 with “appropriate privileges” shall always restore the ownership and permissions of extracted

100257 files exactly as archived. If viewed from the historic equivalence between superuser and  
 100258 “appropriate privileges”, there are two problems with this requirement. First, users running as  
 100259 superusers may unknowingly set dangerous permissions on extracted files. Second, it is  
 100260 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the  
 100261 archive was created by the superuser. (It should be noted that restoration of ownerships and  
 100262 permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to  
 100263 avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the **-p**  
 100264 option. Only a *pax* invocation with the privileges needed, and which has the **-p** option set using  
 100265 the **e** specification character, has appropriate privileges to restore full ownership and permission  
 100266 information.

100267 Note also that this volume of POSIX.1-2008 requires that the file ownership and access  
 100268 permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided  
 100269 with the mode stored in the archive. This means that the file creation mask of the user is applied  
 100270 to the file permissions.

100271 Users should note that directories may be created by *pax* while extracting files with permissions  
 100272 that are different from those that existed at the time the archive was created. When extracting  
 100273 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set  
 100274 their file creation mask appropriately to protect these files during extraction.

100275 The table of contents output is written to standard output to facilitate pipeline processing.

100276 An early proposal had hard links displaying for all pathnames. This was removed because it  
 100277 complicates the output of the case where **-v** is not specified and does not match historical *cpio*  
 100278 usage. The hard-link information is available in the **-v** display.

100279 The description of the **-l** option allows implementations to make hard links to symbolic links.  
 100280 POSIX.1-2008 does not specify any way to create a hard link to a symbolic link, but many  
 100281 implementations provide this capability as an extension. If there are hard links to symbolic links  
 100282 when an archive is created, the implementation is required to archive the hard link in the archive  
 100283 (unless **-H** or **-L** is specified). When in **read** mode and in **copy** mode, implementations  
 100284 supporting hard links to symbolic links should use them when appropriate.

100285 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have  
 100286 been brought along from historical usage. For example, there are restrictions on the length of  
 100287 pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory  
 100288 hierarchies), the ability to use extensions from the **-xpax** format overcomes these restrictions.

100289 The default *blocksize* value of 5120 bytes for *cpio* was selected because it is one of the standard  
 100290 block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value  
 100291 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10240  
 100292 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The  
 100293 maximum block size of 32256 bytes ( $2^{15}$ –512 bytes) is the largest multiple of 512 bytes that fits  
 100294 into a signed 16-bit tape controller transfer register. There are known limitations in some  
 100295 historical systems that would prevent larger blocks from being accepted. Historical values were  
 100296 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate  
 100297 archives. Also, default block sizes for any file type other than character special file has been  
 100298 deleted from this volume of POSIX.1-2008 as unimportant and not likely to affect the structure of  
 100299 the resulting archive.

100300 Implementations are permitted to modify the block-size value based on the archive format or the  
 100301 device to which the archive is being written. This is to provide implementations with the  
 100302 opportunity to take advantage of special types of devices, and it should not be used without a  
 100303 great deal of consideration as it almost certainly decreases archive portability.

- 100304 The intended use of the `-n` option was to permit extraction of one or more files from the archive  
 100305 without processing the entire archive. This was viewed by the standard developers as offering  
 100306 significant performance advantages over historical implementations. The `-n` option in early  
 100307 proposals had three effects; the first was to cause special characters in patterns to not be treated  
 100308 specially. The second was to cause only the first file that matched a pattern to be extracted. The  
 100309 third was to cause *pax* to write a diagnostic message to standard error when no file was found  
 100310 matching a specified pattern. Only the second behavior is retained by this volume of  
 100311 POSIX.1-2008, for many reasons. First, it is in general not acceptable for a single option to have  
 100312 multiple effects. Second, the ability to make pattern matching characters act as normal characters  
 100313 is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the  
 100314 special characters is useful because users may wish to normalize only a single special character  
 100315 in a single filename. Fourth, given a more general escape mechanism, the previous behavior of  
 100316 the `-n` option can be easily obtained using the `-s` option or a *sed* script. Finally, writing a  
 100317 diagnostic message when a pattern specified by the user is unmatched by any file is useful  
 100318 behavior in all cases.
- 100319 In this version, the `-n` was removed from the **copy** mode synopsis of *pax*; it is inapplicable  
 100320 because there are no pattern operands specified in this mode.
- 100321 There is another method than *pax* for copying subtrees in POSIX.1-2008 described as part of the  
 100322 *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface,  
 100323 while *pax* offers a finer granularity of control. Each provides additional functionality to the  
 100324 other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is  
 100325 the intention of the standard developers that the results be similar (using appropriate option  
 100326 combinations in both utilities). The results are not required to be identical; there seemed  
 100327 insufficient gain to applications to balance the difficulty of implementations having to guarantee  
 100328 that the results would be exactly identical.
- 100329 A single archive may span more than one file. It is suggested that implementations provide  
 100330 informative messages to the user on standard error whenever the archive file is changed.
- 100331 The `-d` option (do not create intermediate directories not listed in the archive) found in early  
 100332 proposals was originally provided as a complement to the historic `-d` option of *cpio*. It has been  
 100333 deleted.
- 100334 The `-s` option in early proposals specified a subset of the substitution command from the *ed*  
 100335 utility. As there was no reason for only a subset to be supported, the `-s` option is now compatible  
 100336 with the current *ed* specification. Since the delimiter can be any non-null character, the following  
 100337 usage with single `<space>` characters is valid:
- ```
100338 pax -s " foo bar " ...
```
- 100339 The `-t` description is worded so as to note that this may cause the access time update caused by  
 100340 some other activity (which occurs while the file is being read) to be overwritten.
- 100341 The default behavior of *pax* with regard to file modification times is the same as historical  
 100342 implementations of *tar*. It is not the historical behavior of *cpio*.
- 100343 Because the `-i` option uses `/dev/tty`, utilities without a controlling terminal are not able to use  
 100344 this option.
- 100345 The `-y` option, found in early proposals, has been deleted because a line containing a single  
 100346 `<period>` for the `-i` option has equivalent functionality. The special lines for the `-i` option (a  
 100347 single `<period>` and the empty line) are historical practice in *cpio*.
- 100348 In early drafts, a `-echarmap` option was included to increase portability of files between systems  
 100349 using different coded character sets. This option was omitted because it was apparent that

100350 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate  
 100351 substitute.

100352 The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however,  
 100353 made it very difficult to create a single archive containing files created using extended characters  
 100354 provided by different locales. This version adds the **hdrcharset** keyword to make it possible to  
 100355 archive files in these cases without dropping files due to translation errors.

100356 Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again  
 100357 can lose information, as the resulting filename might not be byte-for-byte equivalent to the  
 100358 original. To avoid this problem, users can specify the **-o hdrcharset=binary** option, which will  
 100359 cause the resulting archive to use binary format for all names and attributes. Such archives are  
 100360 not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based  
 100361 encodings), but they will allow interchange among the vast majority of POSIX file systems in  
 100362 practical use. Also, the **-o hdrcharset=binary** option will cause *pax* in **copy** mode to behave  
 100363 more like other standard utilities such as *cp*.

100364 If the values specified by the **-o exthdr.name=value**, **-o globexthdr.name=value**, or by  
 100365 **\$TMPDIR** (if **-o globexthdr.name** is not specified) require a character encoding other than that  
 100366 described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be  
 100367 created for the file. If a **hdrcharset** extended header record is active for such headers, it will  
 100368 determine the codeset used for the value field in these extended **path** header records. These **path**  
 100369 extended header records always need to be created when writing an archive even if  
 100370 **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in  
 100371 the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record  
 100372 is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even  
 100373 when **hdrcharset=binary** is also in effect for that file.)

100374 The **-k** option was added to address international concerns about the dangers involved in the  
 100375 character set transformations of **-e** (if the target character set were different from the source, the  
 100376 filenames might be transformed into names matching existing files) and also was made more  
 100377 general to protect files transferred between file systems with different {NAME\_MAX} values  
 100378 (truncating a filename on a smaller system might also inadvertently overwrite existing files). As  
 100379 stated, it prevents any overwriting, even if the target file is older than the source. This version  
 100380 adds more granularity of options to solve this problem by introducing the **-o invalid=option**—  
 100381 specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting  
 100382 in this case. The **-k** option closes that loophole.)

100383 Some of the file characteristics referenced in this volume of POSIX.1-2008 might not be  
 100384 supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the  
 100385 file access time. For this reason, the **e** specification character has been provided, intended to  
 100386 cause all file characteristics specified in the archive to be retained.

100387 It is required that extracted directories, by default, have their access and modification times and  
 100388 permissions set to the values specified in the archive. This has obvious problems in that the  
 100389 directories are almost certainly modified after being extracted and that directory permissions  
 100390 may not permit file creation. One possible solution is to create directories with the mode  
 100391 specified in the archive, as modified by the *umask* of the user, with sufficient permissions to  
 100392 allow file creation. After all files have been extracted, *pax* would then reset the access and  
 100393 modification times and permissions as necessary.

100394 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.  
 100395 However, since there is no separate operand list to get conversion arguments, the format was  
 100396 extended to allow specifying the name of the conversion argument as part of the conversion  
 100397 specification.

100398 The T conversion specifier allows time fields to be displayed in any of the date formats. Unlike  
 100399 the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past.  
 100400 This makes parsing the output more predictable.

100401 The D conversion specifier handles the ability to display the major/minor or file size, as with *ls*,  
 100402 by using `%-8 (size)D`.

100403 The L conversion specifier handles the *ls* display for symbolic links.

100404 Conversion specifiers were added to generate existing known types used for *ls*.

### 100405 **pax Interchange Format**

100406 The new POSIX data interchange format was developed primarily to satisfy international  
 100407 concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded  
 100408 in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers  
 100409 realized that this new POSIX data interchange format should be very extensible because there  
 100410 were other requirements they foresaw in the near future:

- 100411 • Support international character encodings and locale information
- 100412 • Support security information (ACLs, and so on)
- 100413 • Support future file types, such as realtime or contiguous files
- 100414 • Include data areas for implementation use
- 100415 • Support systems with words larger than 32 bits and timers with subsecond granularity

100416 The following were not goals for this format because these are better handled by separate  
 100417 utilities or are inappropriate for a portable format:

- 100418 • Encryption
- 100419 • Compression
- 100420 • Data translation between locales and codesets
- 100421 • *inode* storage

100422 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats  
 100423 previously available, only the **ustar** format was selected for extensions because:

- 100424 • It was easier to extend in an upwards-compatible way. It offered version flags and header  
 100425 block type fields with room for future standardization. The **cpio** format, while possessing a  
 100426 more flexible file naming methodology, could not be extended without breaking some  
 100427 theoretical implementation or using a dummy filename that could be a legitimate filename.
- 100428 • Industry experience since the original “*tar wars*” fought in developing the ISO POSIX-1  
 100429 standard has clearly been in favor of the **ustar** format, which is generally the default  
 100430 output format selected for *pax* implementations on new systems.

100431 The new format was designed with one additional goal in mind: reasonable behavior when an  
 100432 older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated  
 100433 that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this  
 100434 allowed the format to include all the extended information in a pseudo-regular file that  
 100435 preceded each real file. An option is given that allows the archive creator to set up reasonable  
 100436 names for these files on the older systems. Also, the normative text suggests that reasonable file  
 100437 access values be used for this **ustar** header block. Making these header files inaccessible for  
 100438 convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are  
 100439 suggested.

- 100440 The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format  
 100441 rather than *magic* or *version* because the POSIX.1-1990 standard (and, by reference, the previous  
 100442 version of *pax*), mandated the behavior of the format-reading utility when it encountered an  
 100443 unknown *typeflag*, but was silent about the other two fields.
- 100444 Early proposals for the first version of this standard contained a proposed archive format that  
 100445 was based on compatibility with the standard for tape files (ISO 1001, similar to the format used  
 100446 historically on many mainframes and minicomputers). This format was overly complex and  
 100447 required considerable overhead in volume and header records. Furthermore, the standard  
 100448 developers felt that it would not be acceptable to the community of POSIX developers, so it was  
 100449 later changed to be a format more closely related to historical practice on POSIX systems.
- 100450 The prefix and name split of pathnames in **ustar** was replaced by the single path extended  
 100451 header record for simplicity.
- 100452 The concept of a global extended header (*typeflagg*) was controversial. If this were applied to an  
 100453 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape  
 100454 could be a serious problem; a utility attempting to extract as many files as possible from a  
 100455 damaged archive could lose a large percentage of file header information in this case. However,  
 100456 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers  
 100457 considerable potential size reductions by eliminating redundant information. Thus, the text  
 100458 warns against using the global method for unreliable media and provides a method for  
 100459 implanting global information in the extended header for each file, rather than in the *typeflag g*  
 100460 records.
- 100461 No facility for data translation or filtering on a per-file basis is included because the standard  
 100462 developers could not invent an interface that would allow this in an efficient manner. If a filter,  
 100463 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the  
 100464 filter to the entire archive as a single file. The standard developers considered interfaces that  
 100465 would invoke a shell script for each file going into or out of the archive, but the system overhead  
 100466 in this approach was considered to be too high.
- 100467 One such approach would be to have **filter=** records that give a pathname for an executable.  
 100468 When the program is invoked, the file and archive would be open for standard input/output  
 100469 and all the header fields would be available as environment variables or command-line  
 100470 arguments. The standard developers did discuss such schemes, but they were omitted from  
 100471 POSIX.1-2008 due to concerns about excessive overhead. Also, the program itself would need to  
 100472 be in the archive if it were to be used portably.
- 100473 There is currently no portable means of identifying the character set(s) used for a file in the file  
 100474 system. Therefore, *pax* has not been given a mechanism to generate charset records  
 100475 automatically. The only portable means of doing this is for the user to write the archive using the  
 100476 **-ocharset=string** command line option. This assumes that all of the files in the archive use the  
 100477 same encoding. The “implementation-defined” text is included to allow for a system that can  
 100478 identify the encodings used for each of its files.
- 100479 The table of standards that accompanies the charset record description is acknowledged to be  
 100480 very limited. Only a limited number of character set standards is reasonable for maximal  
 100481 interchange. Any character set is, of course, possible by prior agreement. It was suggested that  
 100482 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal  
 100483 standards, and then only those with reasonably large followings, can be included here, simply as  
 100484 a matter of practicality. The *<value>*s represent names of officially registered character sets in the  
 100485 format required by the ISO 2375: 1985 standard.
- 100486 The normal *<comma>* or *<blank>*-separated list rules are not followed in the case of keyword  
 100487 options to allow ease of argument parsing for *getopts*.

- 100488 Further information on character encodings is in [pax Archive Character Set Encoding/Decoding](#)  
100489 (on page 3038).
- 100490 The standard developers have reserved keyword name space for vendor extensions. It is  
100491 suggested that the format to be used is:
- 100492 *VENDOR.keyword*
- 100493 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further  
100494 suggested that the keyword following the <period> be named differently than any of the  
100495 standard keywords so that it could be used for future standardization, if appropriate, by  
100496 omitting the *VENDOR* prefix.
- 100497 The <length> field in the extended header record was included to make it simpler to step  
100498 through the records, even if a record contains an unknown format (to a particular *pax*) with  
100499 complex interactions of special characters. It also provides a minor integrity checkpoint within  
100500 the records to aid a program attempting to recover files from a damaged archive.
- 100501 There are no extended header versions of the *devmajor* and *devminor* fields because the  
100502 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific  
100503 extended keywords (such as *VENDOR.devmajor*) should be used.
- 100504 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly  
100505 on a symbolic name basis, as in **ustar**.
- 100506 Just as with the **ustar** format descriptions, the new format makes no special arrangements for  
100507 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file  
100508 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing  
100509 their labels, and mounting each in the proper sequence are considered to be implementation  
100510 details that cannot be described portably.
- 100511 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.  
100512 It is not as densely packed as might be possible for backup:
- 100513 • It contains information as coded characters that could be coded in binary.
  - 100514 • It identifies extended records with name fields that could be omitted in favor of a fixed-  
100515 field layout.
  - 100516 • It translates names into a portable character set and identifies locale-related information,  
100517 both of which are probably unnecessary for backup.
- 100518 The requirements on restoring from an archive are slightly different from the historical wording,  
100519 allowing for non-monolithic privilege to bring forward as much as possible. In particular,  
100520 attributes such as “high performance file” might be broadly but not universally granted while  
100521 set-user-ID or *chown()* might be much more restricted. There is no implication in POSIX.1-2008  
100522 that the security information be honored after it is restored to the file hierarchy, in spite of what  
100523 might be improperly inferred by the silence on that topic. That is a topic for another standard.
- 100524 Links are recorded in the fashion described here because a link can be to any file type. It is  
100525 desirable in general to be able to restore part of an archive selectively and restore all of those files  
100526 completely. If the data is not associated with each link, it is not possible to do this. However, the  
100527 data associated with a file can be large, and when selective restoration is not needed, this can be  
100528 a significant burden. The archive is structured so that files that have no associated data can  
100529 always be restored by the name of any link name of any link, and the user may choose whether  
100530 data is recorded with each instance of a file that contains data. The format permits mixing of  
100531 both types of links in a single archive; this can be done for special needs, and *pax* is expected to  
100532 interpret such archives on input properly, despite the fact that there is no *pax* option that would

100533 force this mixed case on output. (When **-o linkdata** is used, the output must contain the  
100534 duplicate data, but the implementation is free to include it or omit it when **-o linkdata** is not  
100535 used.)

100536 The time values are included as extended header records for those implementations needing  
100537 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be  
100538 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject  
100539 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a  
100540 leading '-'. Even though some implementations can support finer file-time granularities than  
100541 seconds, the normative text requires support only for seconds since the Epoch because the  
100542 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new  
100543 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will  
100544 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification  
100545 time) is described with appropriate privileges so that it can be ignored when writing to the file  
100546 system. POSIX does not provide a portable means to change file creation time. Nothing is  
100547 intended to prevent a non-portable implementation of *pax* from restoring the value.

100548 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the  
100549 sizes specified in the regular *tar* header. New file system architectures are emerging that will  
100550 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits  
100551 for user and group IDs, but the extended header values were included for completeness,  
100552 allowing overrides for all of the decimal values in the *tar* header.

100553 The standard developers intended to describe the effective results of *pax* with regard to file  
100554 ownerships and permissions; implementations are not restricted in timing or sequencing the  
100555 restoration of such, provided the results are as specified.

100556 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The  
100557 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the  
100558 copied files were written to an archive file and then subsequently extracted ...”. There is  
100559 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,  
100560 but the effects must be as if it had.

#### 100561 **pax Archive Character Set Encoding/Decoding**

100562 There is a need to exchange archives of files between systems of different native codesets.  
100563 Filenames, group names, and user names must be preserved to the fullest extent possible when  
100564 an archive is read on the receiving platform. Translation of the contents of files is not within the  
100565 scope of the *pax* utility.

100566 There will also be the need to represent characters that are not available on the receiving  
100567 platform. These unsupported characters cannot be automatically folded to the local set of  
100568 characters due to the chance of collisions. This could result in overwriting previous extracted  
100569 files from the archive or pre-existing files on the system.

100570 For these reasons, the codeset used to represent characters within the extended header records of  
100571 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields  
100572 requiring translation include, at a minimum, filenames, user names, group names, and link  
100573 pathnames. Implementations may wish to have localized extended keywords that use non-  
100574 portable characters.

100575 The standard developers considered the following options:

- 100576 • The archive creator specifies the well-defined name of the source codeset. The receiver  
100577 must then recognize the codeset name and perform the appropriate translations to the  
100578 destination codeset.

- 100579 • The archive creator includes within the archive the character mapping table for the source
- 100580 codeset used to encode extended header records. The receiver must then read the
- 100581 character mapping table and perform the appropriate translations to the destination
- 100582 codeset.
- 100583 • The archive creator translates the extended header records in the source codeset into a
- 100584 canonical form. The receiver must then perform the appropriate translations to the
- 100585 destination codeset.

100586 The approach that incorporates the name of the source codeset poses the problem of codeset  
 100587 name registration, and makes the archive useless to *pax* archive decoders that do not recognize  
 100588 that codeset.

100589 Because parts of an archive may be corrupted, the standard developers felt that including the  
 100590 character map of the source codeset was too fragile. The loss of this one key component could  
 100591 result in making the entire archive useless. (The difference between this and the global extended  
 100592 header decision was that the latter has a workaround—duplicating extended header records on  
 100593 unreliable media—but this would be too burdensome for large character set maps.)

100594 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the  
 100595 cross-product of all source and destination codesets.

100596 To simplify the translation from the source codeset to the canonical form and from the canonical  
 100597 form to the destination codeset, the standard developers decided that the internal representation  
 100598 should be a stateless encoding. A stateless encoding is one where each codepoint has the same  
 100599 meaning, without regard to the decoder being in a specific state. An example of a stateful  
 100600 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the  
 100601 ISO/IEC 646: 1991 standard (equivalent to 7-bit ASCII).

100602 For these reasons, the standard developers decided to adopt a canonical format for the  
 100603 representation of file information strings. The obvious, well-endorsed candidate is the  
 100604 ISO/IEC 10646-1: 2000 standard (based in part on Unicode), which can be used to represent  
 100605 the characters of virtually all standardized character sets. The standard developers initially agreed  
 100606 upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters  
 100607 provides a sufficiently rich set to represent all commonly-used codesets.

100608 However, the standard developers found that the 16-bit Unicode representation had some  
 100609 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character  
 100610 made the extended header records twice as long for the case of strings coded entirely from  
 100611 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the  
 100612 ISO/IEC 10646-1: 2000 standard. This multi-byte representation encodes UCS2 or UCS4  
 100613 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In  
 100614 addition, NUL octets and other characters possibly confusing to POSIX file systems do not  
 100615 appear, except to represent themselves. It was realized that certain national codesets take up  
 100616 more space after the encoding, due to their placement within the UCS range; it was felt that the  
 100617 usefulness of the encoding of the names outweighs the disadvantage of size increase for file,  
 100618 user, and group names.

100619 The encoding of UTF-8 is as follows:

| 100620 | UCS4 Hex Encoding   | UTF-8 Binary Encoding                        |
|--------|---------------------|----------------------------------------------|
| 100621 | 00000000–0000007F   | 0xxxxxxx                                     |
| 100622 | 00000080–000007FF   | 110xxxxx 10xxxxxx                            |
| 100623 | 00000800–0000FFFF   | 1110xxxx 10xxxxxx 10xxxxxx                   |
| 100624 | 00010000–001FFFFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 100625 | 00200000–03FFFFFFF  | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

100626 04000000-7FFFFFFF 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx  
 100627 where each 'x' represents a bit value from the character being translated.

#### 100628 **ustar Interchange Format**

100629 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of  
 100630 the historical *tar* utility. The goal of these changes was not only to provide the functional  
 100631 enhancements desired, but also to retain compatibility between new and old versions. This  
 100632 compatibility has been retained. Archives written using the old archive format are compatible  
 100633 with the new format.

100634 Implementors should be aware that the previous file format did not include a mechanism to  
 100635 archive directory type files. For this reason, the convention of using a filename ending with  
 100636 <slash> was adopted to specify a directory on the archive.

100637 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for  
 100638 {PATH\_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname  
 100639 be stored there without the use of the *prefix* field. Although the *name* field is known to be too  
 100640 small to contain {PATH\_MAX} characters, the value was not changed in this version of the  
 100641 archive file format to retain backwards-compatibility, and instead the *prefix* was introduced.  
 100642 Also, because of the earlier version of the format, there is no way to remove the restriction on the  
 100643 *linkname* field being limited in size to just that of the *name* field.

100644 The *size* field is required to be meaningful in all implementation extensions, although it could be  
 100645 zero. This is required so that the data blocks can always be properly counted.

100646 It is suggested that if device special files need to be represented that cannot be represented in the  
 100647 standard format, that one of the extension types (A-Z) be used, and that the additional  
 100648 information for the special file be represented as data and be reflected in the *size* field.

100649 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with  
 100650 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*  
 100651 should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is  
 100652 converted to another type or not). If run as a privileged user, it should be able to do so, and it  
 100653 would be considered a bug if it did not. The same is true of ordinary data files and similarly  
 100654 named special files; it is impossible to anticipate the needs of the user (who could really intend  
 100655 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the  
 100656 protection system as required.

100657 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a  
 100658 **ustar** archive. POSIX.1-2008 does not require the contiguous file extension, but does define a  
 100659 standard way of archiving such files so that all conforming systems can interpret these file types  
 100660 in a meaningful and consistent manner. On a system that does not support extended file types,  
 100661 the *pax* utility should do the best it can with the file and go on to the next.

100662 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond  
 100663 the usage in the ISO POSIX-2 standard to support the "shared text" or "sticky" bit. It is intended  
 100664 that the conformance document should not document anything beyond the existence of and  
 100665 support of such a mode. Further extensions are expected to these bits, particularly with  
 100666 overloading the set-user-ID and set-group-ID flags.

100667 **cpio Interchange Format**

100668 The reference to appropriate privileges in the **cpio** format refers to an error on standard output;  
100669 the **ustar** format does not make comparable statements.

100670 The model for this format was the historical System V *cpio-c* data interchange format. This  
100671 model documents the portable version of the **cpio** format and not the binary version. It has the  
100672 flexibility to transfer data of any type described within POSIX.1-2008, yet is extensible to transfer  
100673 data types specific to extensions beyond POSIX.1-2008 (for example, contiguous files). Because it  
100674 describes existing practice, there is no question of maintaining upwards-compatibility.

100675 **cpio Header**

100676 There has been some concern that the size of the *c\_ino* field of the header is too small to handle  
100677 those systems that have very large *inode* numbers. However, the *c\_ino* field in the header is used  
100678 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as  
100679 the *inode* number of the file in the location from which that file is extracted.

100680 The name *c\_magic* is based on historical usage.

100681 **cpio Filename**

100682 For most historical implementations of the *cpio* utility, {PATH\_MAX} octets can be used to  
100683 describe the pathname without the addition of any other header fields (the NUL character  
100684 would be included in this count). {PATH\_MAX} is the minimum value for pathname size,  
100685 documented as 256 bytes. However, an implementation may use *c\_namesize* to determine the  
100686 exact length of the pathname. With the current description of the **<cpio.h>** header, this  
100687 pathname size can be as large as a number that is described in six octal digits.

100688 Two values are documented under the *c\_mode* field values to provide for extensibility for known  
100689 file types:

100690 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the  
100691 information for this archive like a regular file. If this file type is undefined, the  
100692 implementation may create the file as a regular file.

100693 This provides for extensibility of the **cpio** format while allowing for the ability to read old  
100694 archives. Files of an unknown type may be read as “regular files” on some implementations. On  
100695 a system that does not support extended file types, the *pax* utility should do the best it can with  
100696 the file and go on to the next.

100697 **FUTURE DIRECTIONS**

100698 None.

100699 **SEE ALSO**

100700 Chapter 2 (on page 2297), *cp*, *ed*, *getopts*, *ls*, *printf*

100701 XBD Section 3.169 (on page 60), Chapter 5 (on page 121), Chapter 8 (on page 173), Section 12.2  
100702 (on page 215), **<cpio.h>**

100703 XSH *chown()*, *creat()*, *fstatat()*, *mkdir()*, *mkfifo()*, *utime()*, *write()*

100704 **CHANGE HISTORY**

100705 First released in Issue 4.

100706 **Issue 5**

100707 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only  
 100708 support files up to 8 gigabytes in size.

100709 **Issue 6**

100710 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 100711 • Support has been added for symbolic links in the options and interchange formats.
- 100712 • A new format has been devised, based on extensions to **ustar**.
- 100713 • References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990  
 100714 standard have been changed to remove the “extended” adjective because this could cause  
 100715 confusion with the extended *tar* header added in this version. (All references to *tar* are  
 100716 actually to **ustar**.)

100717 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

100718 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can  
 100719 ignore an [EEXIST] error when extracting an archive.

100720 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when  
 100721 in **read** mode.

100722 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

100723 IEEE PASC Interpretation 1003.2 #195 is applied.

100724 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**,  
 100725 and **-I** options.

100726 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of  
 100727 the *pax* process into certain fields. This change provides a method for the implementation to  
 100728 ensure that different instances of *pax* extracting a file named */a/b/foo* will not collide when  
 100729 processing the extended header information associated with **foo**.

100730 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **-x B** to **-x pax** in  
 100731 the OPTIONS section.

100732 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to  
 100733 be consistent with the normative text.

100734 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the  
 100735 DESCRIPTION to describe the behavior when files to be linked are symbolic links and the  
 100736 system is not capable of making hard links to symbolic links.

100737 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS  
 100738 section to describe the behavior for how multiple **-odelete=pattern** options are to be handled.

100739 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option  
 100740 within the OPTIONS section.

100741 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into  
 100742 the OPTIONS section that states that specifying more than one of the mutually-exclusive options  
 100743 (**-H** and **-L**) is not considered an error and that the last option specified will determine the  
 100744 behavior of the utility.

100745 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime*  
 100746 paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of  
 100747 the *ctime* keyword for the *pax* extended header, in that the *st\_ctime* member of the **stat** structure  
 100748 does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>**

- 100749 includes a file creation time.
- 100750 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag*
- 100751 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that
- 100752 are aliased via symbolic links.
- 100753 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c\_nlink*
- 100754 field.
- 100755 **Issue 7**
- 100756 Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied.
- 100757 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
- 100758 *LC\_MESSAGES* environment variable.
- 100759 SD5-XCU-ERN-2 is applied, making *-c* and *-n* mutually-exclusive in the SYNOPSIS.
- 100760 SD5-XCU-ERN-3 is applied, revising the default behavior of *-H* and *-L*.
- 100761 SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.
- 100762 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 100763 The *pax* utility is no longer allowed to create separate identical symbolic links when extracting
- 100764 linked symbolic links from an archive.

## 100765 NAME

100766 pr — print files

## 100767 SYNOPSIS

100768 pr [+page] [-column] [-adFmrt] [-e[char][gap]] [-h header] [-i[char][gap]]  
 100769 xSI [-l lines] [-n[char][width]] [-o offset] [-s[char]] [-w width] [-fp]  
 100770 [file...]

## 100771 DESCRIPTION

100772 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be  
 100773 read, formatted, and written to standard output. By default, the input shall be separated into  
 100774 66-line pages, each with:

- 100775 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 100776 • A 5-line trailer consisting of blank lines

100777 If standard output is associated with a terminal, diagnostic messages shall be deferred until the  
 100778 *pr* utility has completed processing.

100779 When options specifying multi-column output are specified, output text columns shall be of  
 100780 equal width; input lines that do not fit into a text column shall be truncated. By default, text  
 100781 columns shall be separated with at least one <blank>.

## 100782 OPTIONS

100783 The *pr* utility shall conform to XBD Section 12.2 (on page 215), except that: the *page* option has a  
 100784 '+' delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are  
 100785 optional; and some of the option-arguments cannot be specified as separate arguments from the  
 100786 preceding option letter. In particular, the *rs* option does not allow the option letter to be  
 100787 separated from its argument, and the options *-e*, *-i*, and *-n* require that both arguments, if  
 100788 present, not be separated from the option letter.

100789 The following options shall be supported. In the following option descriptions, *column*, *lines*,  
 100790 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

100791 *+page* Begin output at page number *page* of the formatted input.

100792 *-column* Produce multi-column output that is arranged in *column* columns (the default shall  
 100793 be 1) and is written down each column in the order in which the text is received  
 100794 from the input file. This option should not be used with *-m*. The options *-e* and *-i*  
 100795 shall be assumed for multiple text-column output. Whether or not text columns are  
 100796 produced with identical vertical lengths is unspecified, but a text column shall  
 100797 never exceed the length of the page (see the *-l* option). When used with *-t*, use the  
 100798 minimum number of lines to write the output.

100799 *-a* Modify the effect of the *-column* option so that the columns are filled across the  
 100800 page in a round-robin order (for example, when *column* is 2, the first input line  
 100801 heads column 1, the second heads column 2, the third is the second line in column  
 100802 1, and so on).

100803 *-d* Produce output that is double-spaced; append an extra <newline> following every  
 100804 <newline> found in the input.

100805 *-e[char][gap]*

100806 Expand each input <tab> to the next greater column position specified by the  
 100807 formula  $n*gap+1$ , where *n* is an integer > 0. If *gap* is zero or is omitted, it shall  
 100808 default to 8. All <tab> characters in the input shall be expanded into the  
 100809 appropriate number of <space> characters. If any non-digit character, *char*, is  
 100810 specified, it shall be used as the input <tab>. If the first character of the *-e* option-

|        |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 100811 |     | argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 100812 | XSI | <b>-f</b> Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline> characters. Pause before beginning the first page if the standard output is associated with a terminal.                                                                                                                                                                                                                                                                                                                                                              |
| 100813 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100814 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100815 |     | <b>-F</b> Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline> characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 100816 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100817 |     | <b>-h header</b> Use the string <i>header</i> to replace the contents of the <i>file</i> operand in the page header.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 100818 |     | <b>-i[<i>char</i>][<i>gap</i>]</b> In output, replace <space> characters with <tab> characters wherever one or more adjacent <space> characters reach column positions <i>gap</i> +1, 2* <i>gap</i> +1, 3* <i>gap</i> +1, and so on. If <i>gap</i> is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, <i>char</i> , is specified, it shall be used as the output <tab>. If the first character of the <b>-i</b> option-argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> . |
| 100819 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100820 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100821 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100822 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100823 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100824 |     | <b>-l lines</b> Override the 66-line default and reset the page length to <i>lines</i> . If <i>lines</i> is not greater than the sum of both the header and trailer depths (in lines), the <i>pr</i> utility shall suppress both the header and trailer, as if the <b>-t</b> option were in effect.                                                                                                                                                                                                                                                                               |
| 100825 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100826 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100827 |     | <b>-m</b> Merge files. Standard output shall be formatted so the <i>pr</i> utility writes one line from each file specified by a <i>file</i> operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine <i>file</i> operands.                                                                                                                                                                                                                                              |
| 100828 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100829 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100830 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100831 |     | <b>-n[<i>char</i>][<i>width</i>]</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 100832 |     | Provide <i>width</i> -digit line numbering (default for <i>width</i> shall be 5). The number shall occupy the first <i>width</i> column positions of each text column of default output or each line of <b>-m</b> output. If <i>char</i> (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for <i>char</i> is a <tab>).                                                                                                                                                                                   |
| 100833 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100834 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100835 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100836 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100837 |     | <b>-o offset</b> Each line of output shall be preceded by offset <space> characters. If the <b>-o</b> option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the <b>-w</b> option below).                                                                                                                                                                                                                                                                                                                        |
| 100838 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100839 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100840 |     | <b>-p</b> Pause before beginning each page if the standard output is directed to a terminal ( <i>pr</i> shall write an <alert> to standard error and wait for a <carriage-return> to be read on /dev/tty).                                                                                                                                                                                                                                                                                                                                                                        |
| 100841 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100842 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100843 |     | <b>-r</b> Write no diagnostic reports on failure to open files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100844 |     | <b>-s[<i>char</i>]</b> Separate text columns by the single character <i>char</i> instead of by the appropriate number of <space> characters (default for <i>char</i> shall be <tab>).                                                                                                                                                                                                                                                                                                                                                                                             |
| 100845 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100846 |     | <b>-t</b> Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.                                                                                                                                                                                                                                                                                                                                                                          |
| 100847 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100848 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100849 |     | <b>-w width</b> Set the width of the line to <i>width</i> column positions for multiple text-column output only. If the <b>-w</b> option is not specified and the <b>-s</b> option is not specified, the default width shall be 72. If the <b>-w</b> option is not specified and the <b>-s</b> option is specified, the default width shall be 512.                                                                                                                                                                                                                               |
| 100850 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100851 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100852 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 100853 |     | For single column output, input lines shall not be truncated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## 100854 OPERANDS

100855 The following operand shall be supported:

100856 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*  
100857 operand is ' - ', the standard input shall be used.

## 100858 STDIN

100859 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '  
100860 See the INPUT FILES section.

## 100861 INPUT FILES

100862 The input files shall be text files.

100863 The file */dev/tty* shall be used to read responses required by the *-p* option.

## 100864 ENVIRONMENT VARIABLES

100865 The following environment variables shall affect the execution of *pr*:

100866 *LANG* Provide a default value for the internationalization variables that are unset or null.  
100867 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
100868 used to determine the values of locale categories.)

100869 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
100870 internationalization variables.

100871 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
100872 characters (for example, single-byte as opposed to multi-byte characters in  
100873 arguments and input files) and which characters are defined as printable (character  
100874 class **print**). Non-printable characters are still written to standard output, but are  
100875 not counted for the purpose for column-width and line-length calculations.

100876 *LC\_MESSAGES*

100877 Determine the locale that should be used to affect the format and contents of  
100878 diagnostic messages written to standard error.

100879 *LC\_TIME* Determine the format of the date and time for use in writing header lines.

100880 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

100881 *TZ* Determine the timezone used to calculate date and time strings written in header  
100882 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

## 100883 ASYNCHRONOUS EVENTS

100884 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error  
100885 messages to the screen before terminating.

## 100886 STDOUT

100887 The *pr* utility output shall be a paginated version of the original file (or files). This pagination  
100888 shall be accomplished using either *<form-feed>* characters or a sequence of *<newline>*  
100889 XSI characters, as controlled by the *-F* or *-f* option. Page headers shall be generated unless the *-t*  
100890 option is specified. The page headers shall be of the form:

100891 "\n\n%s %s Page %d\n\n\n", *<output of date>*, *<file>*, *<page number>*

100892 In the POSIX locale, the *<output of date>* field, representing the date and time of last modification  
100893 of the input file (or the current date and time if the input file is standard input), shall be  
100894 equivalent to the output of the following command as it would appear if executed at the given  
100895 time:

100896 date "+%b %e %H:%M %Y"

- 100897 without the trailing <newline>, if the page being written is from standard input. If the page  
 100898 being written is not from standard input, in the POSIX locale, the same format shall be used, but  
 100899 the time used shall be the modification time of the file corresponding to *file* instead of the current  
 100900 time. When the *LC\_TIME* locale category is not set to the POSIX locale, a different format and  
 100901 order of presentation of this field may be used.
- 100902 If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null  
 100903 string.
- 100904 If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.
- 100905 **STDERR**
- 100906 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**  
 100907 is specified.
- 100908 **OUTPUT FILES**
- 100909 None.
- 100910 **EXTENDED DESCRIPTION**
- 100911 None.
- 100912 **EXIT STATUS**
- 100913 The following exit values shall be returned:
- 100914 0 Successful completion.
- 100915 >0 An error occurred.
- 100916 **CONSEQUENCES OF ERRORS**
- 100917 Default.
- 100918 **APPLICATION USAGE**
- 100919 A conforming application must protect its first operand, if it starts with a <plus-sign>, by  
 100920 preceding it with the "--" argument that denotes the end of the options. For example, *pr+x*  
 100921 could be interpreted as an invalid page number or a *file* operand.
- 100922 **EXAMPLES**
- 100923 1. Print a numbered list of all files in the current directory:
- 100924 `ls -a | pr -n -h "Files in $(pwd)."`
- 100925 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":
- 100926 `pr -3d -h "file list" file1 file2`
- 100927 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:
- 100928 `pr -e9 -t <file1 >file2`
- 100929 **RATIONALE**
- 100930 This utility is one of those that does not follow the Utility Syntax Guidelines because of its  
 100931 historical origins. The standard developers could have added new options that obeyed the  
 100932 guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are  
 100933 examples of both actions in this volume of POSIX.1-2008. Because of its widespread use by  
 100934 historical applications, the standard developers decided to exempt this version of *pr* from many  
 100935 of the guidelines.
- 100936 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options  
 100937 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by  
 100938 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical

- 100939 practice because they are frequently specified without their optional arguments. If a <blank>  
 100940 were allowed before the option-argument in these cases, a *file* operand could mistakenly be  
 100941 interpreted as an option-argument in historical applications.
- 100942 The text about the minimum number of lines in multi-column output was included to ensure  
 100943 that a best effort is made in balancing the length of the columns. There are known historical  
 100944 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines  
 100945 and a second of 4. Although this is not a problem when a full page with headers and trailers is  
 100946 produced, it would be relatively useless when used with *-t*.
- 100947 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.  
 100948 BSD uses it as described here for the *-F* option; System V uses it to change trailing <newline>  
 100949 characters on each page to a <form-feed> and, if standard output is a TTY device, sends an  
 100950 <alert> to standard error and reads a line from */dev/tty* before the first page. There were strong  
 100951 arguments from both sides of this issue concerning historical practice and as a result the *-F*  
 100952 option was added. XSI-conformant systems support the System V historical actions for the *-f*  
 100953 option.
- 100954 The <*output of date*> field in the *-l* format is specified only for the POSIX locale. As noted, the  
 100955 format can be different in other locales. No mechanism for defining this is present in this volume  
 100956 of POSIX.1-2008, as the appropriate vehicle is a message catalog; that is, the format should be  
 100957 specified as a “message”.
- 100958 **FUTURE DIRECTIONS**
- 100959 None.
- 100960 **SEE ALSO**
- 100961 *expand*, *lp*
- 100962 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 100963 **CHANGE HISTORY**
- 100964 First released in Issue 2.
- 100965 **Issue 6**
- 100966 The following new requirements on POSIX implementations derive from alignment with the  
 100967 Single UNIX Specification:
- 100968 • The *-p* option is added.
- 100969 The normative text is reworded to avoid use of the term “must” for application requirements.
- 100970 **Issue 7**
- 100971 PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.
- 100972 Austin Group Interpretation 1003.1-2001 #093 is applied.
- 100973 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100974 **NAME**

100975 printf — write formatted output

100976 **SYNOPSIS**100977 printf *format* [*argument...*]100978 **DESCRIPTION**100979 The *printf* utility shall write formatted operands to the standard output. The *argument* operands  
100980 shall be formatted under control of the *format* operand.100981 **OPTIONS**

100982 None.

100983 **OPERANDS**

100984 The following operands shall be supported:

100985 *format* A string describing the format to use to write the remaining operands. See the  
100986 EXTENDED DESCRIPTION section.100987 *argument* The strings to be written to standard output, under the control of *format*. See the  
100988 EXTENDED DESCRIPTION section.100989 **STDIN**

100990 Not used.

100991 **INPUT FILES**

100992 None.

100993 **ENVIRONMENT VARIABLES**100994 The following environment variables shall affect the execution of *printf*:100995 *LANG* Provide a default value for the internationalization variables that are unset or null.  
100996 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
100997 used to determine the values of locale categories.)100998 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
100999 internationalization variables.101000 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
101001 characters (for example, single-byte as opposed to multi-byte characters in  
101002 arguments).101003 *LC\_MESSAGES*  
101004 Determine the locale that should be used to affect the format and contents of  
101005 diagnostic messages written to standard error.101006 *LC\_NUMERIC*  
101007 Determine the locale for numeric formatting. It shall affect the format of numbers  
101008 written using the *e*, *E*, *f*, *g*, and *G* conversion specifier characters (if supported).101009 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.101010 **ASYNCHRONOUS EVENTS**

101011 Default.

101012 **STDOUT**

101013 See the EXTENDED DESCRIPTION section.

**printf**101014 **STDERR**

101015 The standard error shall be used only for diagnostic messages.

101016 **OUTPUT FILES**

101017 None.

101018 **EXTENDED DESCRIPTION**

101019 The *format* operand shall be used as the *format* string described in XBD Chapter 5 (on page 121)  
101020 with the following exceptions:

- 101021 1. A <space> in the format string, in any context other than a flag of a conversion  
101022 specification, shall be treated as an ordinary character that is copied to the output.
- 101023 2. A 'Δ' character in the format string shall be treated as a 'Δ' character, not as a <space>.
- 101024 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 121) ('\\', '\a',  
101025 '\b', '\f', '\n', '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit  
101026 octal number, shall be written as a byte with the numeric value specified by the octal  
101027 number.
- 101028 4. The implementation shall not precede or follow output from the *d* or *u* conversion  
101029 specifiers with <blank> characters not specified by the *format* operand.
- 101030 5. The implementation shall not precede output from the *o* conversion specifier with zeros  
101031 not specified by the *format* operand.
- 101032 6. The *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers need not be supported.
- 101033 7. An additional conversion specifier character, *b*, shall be supported as follows. The  
101034 argument shall be taken to be a string that may contain <backslash>-escape sequences.  
101035 The following <backslash>-escape sequences shall be supported:
  - 101036 — The escape sequences listed in XBD Chapter 5 (on page 121) ('\\', '\a', '\b',  
101037 '\f', '\n', '\r', '\t', '\v'), which shall be converted to the characters they  
101038 represent
  - 101039 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be  
101040 converted to a byte with the numeric value specified by the octal number
  - 101041 — '\c', which shall not be written and shall cause *printf* to ignore any remaining  
101042 characters in the string operand containing it, any remaining string operands, and  
101043 any additional characters in the *format* operand
- 101044 The interpretation of a <backslash> followed by any other sequence of characters is  
101045 unspecified.
- 101046 Bytes from the converted string shall be written until the end of the string or the number  
101047 of bytes indicated by the precision specification is reached. If the precision is omitted, it  
101048 shall be taken to be infinite, so all bytes up to the end of the converted string shall be  
101049 written.
- 101050 8. For each conversion specification that consumes an argument, the next argument operand  
101051 shall be evaluated and converted to the appropriate type for the conversion as specified  
101052 below.
- 101053 9. The *format* operand shall be reused as often as necessary to satisfy the argument  
101054 operands. Any extra *c* or *s* conversion specifiers shall be evaluated as if a null string  
101055 argument were supplied; other extra conversion specifications shall be evaluated as if a  
101056 zero argument were supplied. If the *format* operand contains no conversion specifications  
101057 and *argument* operands are present, the results are unspecified.

- 101058 10. If a character sequence in the *format* operand begins with a '%' character, but does not  
101059 form a valid conversion specification, the behavior is unspecified.
- 101060 11. The argument to the *c* conversion specifier can be a string containing zero or more bytes.  
101061 If it contains one or more bytes, the first byte shall be written and any additional bytes  
101062 shall be ignored. If the argument is an empty string, it is unspecified whether nothing is  
101063 written or a null byte is written.

101064 The *argument* operands shall be treated as strings if the corresponding conversion specifier is *b*,  
101065 *c*, or *s*, and shall be evaluated as if by the *strtod()* function if the corresponding conversion  
101066 specifier is *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G*. Otherwise, they shall be evaluated as unsuffixed C integer  
101067 constants, as described by the ISO C standard, with the following extensions:

- 101068 • A leading <plus-sign> or minus-sign shall be allowed.
- 101069 • If the leading character is a single-quote or double-quote, the value shall be the numeric  
101070 value in the underlying codeset of the character following the single-quote or double-  
101071 quote.
- 101072 • Suffixed integer constants may be allowed.

101073 If an argument operand cannot be completely converted into an internal value appropriate to  
101074 the corresponding conversion specification, a diagnostic message shall be written to standard  
101075 error and the utility shall not exit with a zero exit status, but shall continue processing any  
101076 remaining operands and shall write the value accumulated at the time the error was detected to  
101077 standard output.

101078 It is not considered an error if an argument operand is not completely used for a *c* or *s*  
101079 conversion.

#### 101080 EXIT STATUS

101081 The following exit values shall be returned:

- 101082 0 Successful completion.
- 101083 >0 An error occurred.

#### 101084 CONSEQUENCES OF ERRORS

101085 Default.

#### 101086 APPLICATION USAGE

101087 The floating-point formatting conversion specifications of *printf()* are not required because all  
101088 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations  
101089 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-  
101090 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility  
101091 cannot really be used to format *bc* output; it does not support arbitrary precision.)  
101092 Implementations are encouraged to support the floating-point conversions as an extension.

101093 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of  
101094 POSIX.1-2008 on which it is based, makes no special provision for dealing with multi-byte  
101095 characters when using the *%c* conversion specification or when a precision is specified in a *%b* or  
101096 *%s* conversion specification. Applications should be extremely cautious using either of these  
101097 features when there are multi-byte characters in the character set.

101098 No provision is made in this volume of POSIX.1-2008 which allows field widths and precisions  
101099 to be specified as '\*' since the '\*' can be replaced directly in the *format* operand using shell  
101100 variable substitution. Implementations can also provide this feature as an extension if they so  
101101 choose.

**printf**

101102 Hexadecimal character constants as defined in the ISO C standard are not recognized in the  
 101103 *format* operand because there is no consistent way to detect the end of the constant. Octal  
 101104 character constants are limited to, at most, three octal digits, but hexadecimal character constants  
 101105 are only terminated by a non-hex-digit character. In the ISO C standard, the "##" concatenation  
 101106 operator can be used to terminate a constant and follow it with a hexadecimal character to be  
 101107 written. In the shell, concatenation occurs before the *printf* utility has a chance to parse the end  
 101108 of the hexadecimal constant.

101109 The %b conversion specification is not part of the ISO C standard; it has been added here as a  
 101110 portable way to process <backslash>-escapes expanded in string operands as provided by the  
 101111 *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2615) for ways to use  
 101112 *printf* as a replacement for all of the traditional versions of the *echo* utility.

101113 If an argument cannot be parsed correctly for the corresponding conversion specification, the  
 101114 *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end  
 101115 of an argument being used for a numeric conversion shall be reported as errors.

**EXAMPLES**

101116 To alert the user and then print and read a series of prompts:

```
101118 printf "\aPlease fill in the following: \nName: "
101119 read name
101120 printf "Phone number: "
101121 read phone
```

101122 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and  
 101123 print them out. The numbers are right-justified and separated by a single <tab>. The percentage  
 101124 is written to one decimal place of accuracy:

```
101125 while read right wrong ; do
101126     percent=$(echo "scale=1; ($right*100)/($right+$wrong)" | bc)
101127     printf "%2d right\t%2d wrong\t(%s%%)\n" \
101128         $right $wrong $percent
101129 done < database_file
```

101130 The command:

```
101131 printf "%5d%4d\n" 1 21 321 4321 54321
```

101132 produces:

```
101133     1  21
101134    321 4321
101135   54321  0
```

101136 Note that the *format* operand is used three times to print all of the given strings and that a '0'  
 101137 was supplied by *printf* to satisfy the last %4d conversion specification.

101138 The *printf* utility is required to notify the user when conversion errors are detected while  
 101139 producing numeric output; thus, the following results would be expected on an implementation  
 101140 with 32-bit twos-complement integers when %d is specified as the *format* operand:

| Argument    | Standard Output | Diagnostic Output                                |
|-------------|-----------------|--------------------------------------------------|
| 5a          | 5               | <b>printf: "5a" not completely converted</b>     |
| 9999999999  | 2147483647      | <b>printf: "9999999999" arithmetic overflow</b>  |
| -9999999999 | -2147483648     | <b>printf: "-9999999999" arithmetic overflow</b> |
| ABC         | 0               | <b>printf: "ABC" expected numeric value</b>      |

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the `strtol()` function as defined in the System Interfaces volume of POSIX.1-2008. A similar correspondence exists between `%u` and `strtoul()` and `%e`, `%f`, and `%g` (if the implementation supports floating-point conversions) and `strtod()`.

In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

```
printf "%d\n" 3 +3 -3 \'3 \' +3 "'-3"
```

produces:

3 Numeric value of constant 3

3 Numeric value of constant 3

-3 Numeric value of constant -3

51 Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset

43 Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset

45 Numeric value of the character '-' in the ISO/IEC 646:1991 standard codeset

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the `wchar_t` representation of the character as described in the System Interfaces volume of POSIX.1-2008.

#### RATIONALE

The `printf` utility was added to provide functionality that has historically been provided by `echo`. However, due to irreconcilable differences in the various versions of `echo` extant, the version has few special features, leaving those to this new `printf` utility, which is based on one in the Ninth Edition system.

The EXTENDED DESCRIPTION section almost exactly matches the `printf()` function in the ISO C standard, although it is described in terms of the file format notation in XBD Chapter 5 (on page 121).

Earlier versions of this standard specified that arguments for all conversions other than `b`, `c`, and `s` were evaluated in the same way (as C constants, but with stated exceptions). For implementations supporting the floating-point conversions it was not clear whether integer conversions need only accept integer constants and floating-point conversions need only accept floating-point constants, or whether both types of conversions should accept both types of constants. Also by not distinguishing between them, the requirement relating to a leading single-quote or double-quote applied to floating-point conversions even though this provided no useful functionality to applications that was not already available through the integer conversions. The current standard clarifies the situation by specifying that the arguments for floating-point conversions are evaluated as if by `strtod()`, and the arguments for integer conversions are evaluated as C integer constants, with the special treatment of leading single-quote and double-quote applying only to integer conversions.

**printf**

Utilities

101184 **FUTURE DIRECTIONS**

101185 None.

101186 **SEE ALSO**101187 *awk, bc, echo*101188 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173)101189 XSH *fprintf()*, *strtod()*101190 **CHANGE HISTORY**

101191 First released in Issue 4.

101192 **Issue 7**

101193 Austin Group Interpretations 1003.1-2001 #175 and #177 are applied.

101194 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

101195 **NAME**101196            *prs* — print an SCCS file (**DEVELOPMENT**)101197 **SYNOPSIS**

```

101198 XSI        prs [-a] [-d dataspec] [-r[SID]] file...
101199            prs [-e|-l] -c cutoff [-d dataspec] file...
101200            prs [-e|-l] -r[SID] [-d dataspec] file...

```

101201 **DESCRIPTION**

101202            The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied  
 101203            format.

101204 **OPTIONS**

101205            The *prs* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the *-r* option has an  
 101206            optional option-argument. This optional option-argument cannot be presented as a separate  
 101207            argument. The following options shall be supported:

101208            *-d dataspec*    Specify the output data specification. The *dataspec* shall be a string consisting of  
 101209            SCCS file *data keywords* (see [Data Keywords](#), on page 3056) interspersed with  
 101210            optional user-supplied text.

101211            *-r[SID]*            Specify the SCCS identification string (*SID*) of a delta for which information is  
 101212            desired. If no *SID* option-argument is specified, the *SID* of the most recently  
 101213            created delta shall be assumed.

101214            *-e*                    Request information for all deltas created earlier than and including the delta  
 101215            designated via the *-r* option or the date-time given by the *-c* option.

101216            *-l*                    Request information for all deltas created later than and including the delta  
 101217            designated via the *-r* option or the date-time given by the *-c* option.

101218            *-c cutoff*            Indicate the *cutoff* date-time, in the form:

101219            *YY[MM[DD[HH[MM[SS]]]]]*

101220            For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999  
 101221            inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

101222            **Note:**            It is expected that in a future version of this standard the default century inferred  
 101223            from a 2-digit year will change. (This would apply to all commands accepting a  
 101224            2-digit year as input.)

101225            No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
 101226            date-time shall be included in the output. Units omitted from the date-time default  
 101227            to their maximum possible values; for example, *-c 7502* is equivalent to  
 101228            *-c 750228235959*.

101229            *-a*                    Request writing of information for both removed—that is, *delta type=R* (see  
 101230            *rmDEL*)—and existing—that is, *delta type=D*,—deltas. If the *-a* option is not  
 101231            specified, information for existing deltas only shall be provided.

101232 **OPERANDS**

101233            The following operand shall be supported:

101234            *file*                A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*  
 101235            utility shall behave as though each file in the directory were specified as a named  
 101236            file, except that non-SCCS files (last component of the pathname does not begin  
 101237            with *s.*) and unreadable files shall be silently ignored.

- 101238 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;  
 101239 each line of the standard input shall be taken to be the name of an SCCS file to be  
 101240 processed. Non-SCCS files and unreadable files shall be silently ignored.
- 101241 **STDIN**
- 101242 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each  
 101243 line of the text file shall be interpreted as an SCCS pathname.
- 101244 **INPUT FILES**
- 101245 Any SCCS files displayed are files of an unspecified format.
- 101246 **ENVIRONMENT VARIABLES**
- 101247 The following environment variables shall affect the execution of *prs*:
- 101248 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 101249 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables  
 101250 used to determine the values of locale categories.)
- 101251 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 101252 internationalization variables.
- 101253 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 101254 characters (for example, single-byte as opposed to multi-byte characters in  
 101255 arguments and input files).
- 101256 *LC\_MESSAGES*  
 101257 Determine the locale that should be used to affect the format and contents of  
 101258 diagnostic messages written to standard error.
- 101259 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 101260 **ASYNCHRONOUS EVENTS**
- 101261 Default.
- 101262 **STDOUT**
- 101263 The standard output shall be a text file whose format is dependent on the data keywords  
 101264 specified with the *-d* option.
- 101265 **Data Keywords**
- 101266 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an  
 101267 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple  
 101268 times.
- 101269 The information written by *prs* shall consist of:
- 101270 1. The user-supplied text
- 101271 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data  
 101272 keywords in the order of appearance in the *dataspec*
- 101273 The format of a data keyword value shall either be simple (*'S'*), in which keyword substitution  
 101274 is direct, or multi-line (*'M'*).
- 101275 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be  
 101276 specified by *'\t'* and <newline> by *'\n'*. When the *-r* option is not specified, the default  
 101277 *dataspec* shall be:
- 101278 :PN: :\n\n
- 101279 and the following *dataspec* shall be used for each selected delta: